BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

W1333

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
22 March 2001 (22.03.2001)

PCT

(10) International Publication Number
WO 01/20535 A2

(51) International Patent Classification[7]: G06F 19/00

(21) International Application Number: PCT/US00/25247

(22) International Filing Date:
14 September 2000 (14.09.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/154,149	14 September 1999 (14.09.1999)	US
09/397,335	14 September 1999 (14.09.1999)	US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier applications:

US	60/154,149 (CIP)
Filed on	14 September 1999 (14.09.1999)
US	09/397,335 (CIP)
Filed on	14 September 1999 (14.09.1999)

(71) Applicant *(for all designated States except US)*: ERAGEN BIOSCIENCES, INC. [US/US]; 12085 Research Drive, Alachua, FL 32615 (US).

(72) Inventors; and

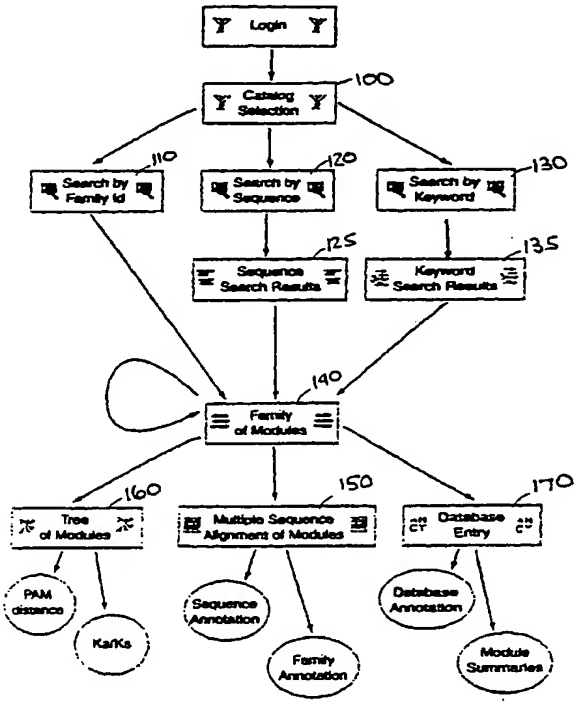
(75) Inventors/Applicants *(for US only)*: CHAMBERLIN, Stephen [GB/US]; 12085 Research Drive, Alachua, FL 32615 (US). BENNER, Steven, A. [US/US]; 1501 NW 68th Terrace, Gainesville, FL 32605 (US). KNECHT, Lukas [CH/US]; Neptunstrasse 35, CH-8032 Zurich (CH).

(74) Agents: McLEOD, Christine, Q. et al.; Saliwanchik, Lloyd & Saliwanchik, 2421 N.W. 41st Street, Suite A-1, Gainesville, FL 32606 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ,

[Continued on next page]

(54) Title: GRAPHICAL USER INTERFACE FOR DISPLAY AND ANALYSIS OF BIOLOGICAL SEQUENCE DATA



(57) Abstract: A computer research tool is provided for searching and displaying biological data. Specifically, the invention provides a computer research tool for performing computerized research of biological data from various databases and for providing a novel graphical user interface that significantly enhances biological data representation, progressive querying and cross-navigation of windows and databases. The invention can be implemented in numerous ways, including as a system, a device, a method, or a computer readable medium.

WO 01/20535 A2

NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) **Designated States** *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

DESCRIPTION

GRAPHICAL USER INTERFACE FOR DISPLAY AND ANALYSIS OF BIOLOGICAL SEQUENCE DATA

Cross-Reference to Related Applications

This application claims priority from United States provisional application, serial number 60/154,149, filed September 14, 1999, and United States patent application serial number 09/397,335, filed September 14. 1999, the disclosures of which are incorporated herein by reference in their entirety.

Technical Field

The present invention relates to a computer research tool for searching and displaying biological data. More specifically, the invention relates to a computer research tool utilizing a novel graphical user interface (GUI) for performing computerized research of biological data from various databases and for providing enhanced graphical representation of biological data, progressive querying, and cross-navigation of relational data.

Background Art

Trillions of pieces of information generated by emerging technologies in molecular biology and genetics are stored digitally in computer databases worldwide. In fact, every day approximately 2000 nucleotide sequences are deposited in publically accessible databases. With such a large amount of multidimensional data, researchers rely on complex information systems to find, summarize and interpret this biological information. This has resulted in the creation of a new field of science known as bioinformatics which combines the power of biochemistry,

mathematics, and computers. Bioinformatics has allowed the development of new databases and computational technologies to help in the understanding of the biological meaning encoded in vast collections of sequence data.

A. Biochemistry Overview

An understanding of the biological meaning of sequence data begins with a study of the 20 amino acids that make up proteins. Deoxyribonucleic acid (DNA) contains the blueprints for these structures. DNA is composed of very long polymers of chemical sub-units known as nucleotides. Each nucleotide includes one of four nitrogenous bases: adenine (A), thymine (T), cytosine (C) and guanine (G). DNA serves as a template for ribonucleic acid (RNA), which serves as a template for proteins. Like DNA, RNA is also composed of nucleotides. Each RNA nucleotide includes one of four nitrogenous bases. These bases of RNA differ from that of DNA only in the substitution of thymine (T) with uracil (U). Three nucleotides of DNA encode three nucleotides of RNA, which in turn encode one amino acid of a protein.

Proteins are macromolecules of amino acids which show great diversity in physical properties thereby fulfilling a broad range of biological functions (i.e., polymers of covalently bonded amino acids). A protein's structure and function depends upon its amino acid sequence, which is determined by the nucleotide sequence of the RNA which produced it, which is determined by the nucleotide sequence of the DNA that produced the RNA. Hence, the great diversity observed in the sequence of amino acids is the direct result of the many possible permutations of DNA and RNA. The primary structure is the sequence of amino acids covalently bonded together. The secondary structure is the result of amino acid sequence of the polypeptide. The bonding causes the chain to develop specific shapes (alpha helix, beta sheet). The tertiary structure is the 3-dimensional folding of the alpha helix or the pleated sheet. The quaternary structure is the spatial relationship between the different polypeptides in the protein.

B. Sequence Comparison

Sequence comparison is a very powerful tool in molecular biology, genetics and protein chemistry. Frequently, it is unknown for which proteins a new DNA sequence codes or if it codes for any protein at all. If you compare a new coding sequence with all known sequences there is a high probability to find a similar sequence. Usually one tries to determine what level of similarity is shared between the proteins in terms of structural and functional characteristics. This determination is made by comparing the amino acid sequences of the proteins. It has been observed that the primary structures of a given protein from related species closely resemble one another. Comparisons of the primary structures of homologous proteins (evolutionarily related proteins) indicate which of the proteins' amino acid residues or domains (i.e., stretches of amino acids) are essential to its function, which are of lesser significance, and which have little specific

function. Sequences which are found in similar positions of functionally similar proteins are said to be homologous, conservatively substituted or highly conserved. A popular computational tool for rapid comparison of a search sequence to a database of known sequences is the BLAST search. The advantage of a BLAST search is the ability to find matches to distantly related sequences. The disadvantage is that the searches become computationally intensive and may take an inordinate length of time.

C. Biological Databases

In order to perform these sequence comparisons, databases of known biological data need to be accessed. There are a lot of different data banks (databases) where biological information such as DNA and protein sequence data are stored, including, general biological databanks such as EMBL/GENBANK (nucleotide sequences), SWISSPROT (protein sequences), and PDB/Protein Data Bank (protein structures). [See, e.g., "Comprehensive, Comprehensible, Distributed and Intelligent Databases: Current Status" by Frishman, et al. *Bioinformatics Review*, Vol. 14, No. 7, 1998, pgs. 551-561, incorporated herein by reference]. Specifically, GenBank is an annotated collection of all publically available DNA sequences. As of August 1999 there were approximately 3,400,000,000 bases in 4,610,000 sequence records. The Genbank database comprises the DNA DataBank of Japan (DDBJ), the European Molecular Biology Laboratory (EMBL), and GenBank at NCBI. SWISS-PROT is an annotated protein sequence database maintained by the Department of Medical Biochemistry of the University of Geneva. The PDB/Protein Data Bank, maintained by Brookhaven National Laboratory, contains all publically available solved protein structures. These databases contain large amounts of raw sequence data which can be cumbersome to use.

In an effort to provide a more useful form of biological data, there are a number of derived or structured databases which integrate information from multiple primary sources, and may include relational/cross-referenced data with respect to sequence, structure, function, and evolution. A derived database generally contains added descriptive materials on top of the primary data or provides novel structuring of the data based on certain defined relationships. Derived/structured databases typically structure the protein sequence data into usable sets of data (tables), grouping the protein sequences by family or by homology domains. A protein family is a group of sequences that can be aligned from end to end and are <55% different globally. A homologous domain is a subsequence of a protein that is distinguished by a well-defined set of properties or characteristics and may also occur in at least two different subfamilies.

An example of structured database is ProDom, a protein domain database, consisting of an automatic compilation of homologous domains. The database was designed as a tool to help

analyze domain arrangements of proteins and protein families. Current versions of the ProDom database are built using a procedure based on recursive PSI-BLAST searches. ProDom contains 57,976 domain families, sorted by decreasing number of protein sequences in the families. ProDom is generated from the SWISS-PROT database by automated sequence comparison.

Similarly, DOMO is a database of homologous protein domain families. DOMO was obtained from successive sequence analysis steps including similarity search, domain delineation, multiple sequence alignment, and motif construction. DOMO has analyzed 83,054 non redundant protein sequences from SWISS-PROT and PIR-International Sequence DataBase yielding a database of 99,058 domain clusters into 8,877 multiple sequence alignments.

Another derived protein sequence database is the Block Database. Blocks are multiply aligned ungapped segments corresponding to the most highly conserved regions of proteins. The blocks for the Block Database are made automatically by looking for the most highly conserved regions in groups of proteins documented in the Prosite Database. The blocks are then calibrated against the SWISS-PROT database to obtain a measure of the chance distribution matches.

D. Researching Biological Databases

Typically biological databases may be searched by either an unstructured (keyword) or structured (field based) search. An unstructured search of the database is preformed by searching for a keyword or the ID of records. For example, a keyword search of "ecoli" retrieves a list of protein sequences that are identified by the keyword "ecoli". A structured search is a more deliberate search, allowing, for example, the searching of the database for protein sequences which contain a particular sequence of interest.

An example of a well known search engine to conduct research on the GenBank database is the ENTREZ search engine which utilizes keyword searching. If a search results in too many hits, ENTREZ allows the addition of new search terms to progressively narrow the number of hits. A researcher may then select all or a subset of the entries that match the search for display to generate a summary page that reports on each of the selected entries. The search results may be displayed in a variety of formats or standardized reports. The form that most biologists are familiar with, the GenBank report, shows the raw GenBank entry. Other familiar formats include FASTA and ASN.1. The genomes division of ENTREZ has a graphic interface based on alignments among multiple maps. The display image shows a series of genetic and physical maps published from a variety of sources, roughly aligned, with diagonal lines connecting common features.

Another search system is SRS (Sequence Retrieval System) which is a Web-based system for searching among multiple sequence databases supported by EMBL. The SRS cross-references sequence information from approximately 40 other sequence databases including ones

that hold protein and nucleotide sequence information, 3D structure, disease and phenotype information, and functional information. The SRS search allows structured queries on one or more databases with common fields (e.g., ID, AccNumber, Description). SRS displays the results as a series of hypertext links. The search can be broadened to other databases by bringing in cross-references.

A number of patents exist which relate to display and analysis of biological sequence data, including, U.S. Patent Nos. 5891632; 5884230; 5878373; 5873052; 5873052; 5864488; 5856928; 5842151; 5799301; 5795716; 5724605; 5724253; 5706498; 5701256; 5600826; 5598350; 5595877; 5577249; 5557535; 5524240; 5453937; 5187775; 4939666; 4923808; 4771384; and 4704692; and PCT Patent No. WO96/23078; all of which are incorporated herein by reference.

E. Graphical User Interfaces (GUIs)

The development and proliferation of GUIs has greatly enhanced the ease with which users interact with biological databases both in the searching stage and in the display of information. A conventional GUI display includes a desktop metaphor upon which one or more icons, application windows, or other graphical objects are displayed. Typically, a data processing system user interacts with a GUI display utilizing a graphical pointer, which the user controls with a graphical pointing device, such as a mouse, trackball, or joystick. For example, depending upon the actions allowed by the active application or operating system software, the user can select icons or other graphical objects within the GUI display by positioning the graphical pointer over the graphical object and depressing a button associated with the graphical pointing device. In addition, the user can typically relocate icons, application windows, and other graphical objects on the desktop utilizing the well known drag-and-drop techniques. By manipulating the graphical objects within the GUI display, the user can control the underlying hardware devices and software objects represented by the graphical objects in a graphical and intuitive manner.

User interfaces used with multi-tasking processors also allow the user to simultaneously work on many tasks at once, each task being confined to its own display window. The interface allows the presentation of multiple windows in potentially overlapping relationships on a display screen. The user can thus retain a window on the screen while temporarily superimposing a further window entirely or partially overlapping the retained window. This enables the user to divert the attention from a first window to one or more secondary windows for assistance and /or references, so that overall user interaction may be improved. There may be many windows with active applications running at once. Oftentimes, the windows may be (dynamically or

statically) related such that modifying a query in one window results in changes to the displayed data in the other related windows, thereby "propagating" the changes throughout.

There are a number of patents which relate to Graphical User Interfaces. For example, the following patents which relate to Graphical User Interfaces, albeit not for biological data, are incorporated by reference herein: U.S. Patent No. 5,926,806 to Marshall et al.; U.S. Patent No. 5,544,352 to Egger; U.S. Patent No. 5,777,616 to Bates et al.; U.S. Patent No. 5,812,804 to Bates et al.; U.S. Patent No. 5,146,556 to Hullot et al.; U.S. Patent No. 5,893,082 to McCormick; U.S. Patent No. 5,815,151 to Argiolas; U.S. Patent No. 5,911,138 to Li; U.S. Patent No. 5,761,656 to Ben-Shachar; U.S. Patent No. 5,404,442 to Foster et al.; U.S. Patent No. 5,917,492 to Bereiter et al.; U.S. Patent No. 4.710,763 to Franke et al.; U.S. Patent No. 5,828,376 to Solimene et al.; U.S. Patent No. 5,748,927 to Stein et al.; U.S. Patent No. 5,452,416 to Hilton et al.; and 5,721,900 to Banning.

F. Graphical User Interfaces for Biological Data Systems

As in most industries, software user interfaces for biological data have evolved from the former DOS text and command line interfaces to intuitive screen graphics which represent data in a user friendly manner. In order to evaluate and analyze data sequences from various biological databases, researchers often utilize graphical user interfaces to view biological data in a variety of ways, including multiple sequence alignments (MSAs), secondary structure predictions, two-dimensional graphical representations of sequences, and phylogenetic trees.

A multiple sequence alignment displays the alignment of homologous residues among a set of sequences in columns. In a 2D graphical representation, sequences are displayed as schematic boxes wherein each box is spatially oriented. Phylogenetic trees are genealogical trees which are built up with information gained from the comparison of the amino acid sequences in a protein. The phylogenetic tree (rooted or unrooted) is a graphical representation of the evolutionary distance between individual protein sequences in a family of proteins. The branches of the phylogenetic tree are evolutionary distances from the PAM matrix, an evolutionary model that assumes that estimation of mutation rates for closely related proteins can be extrapolated to distant relationships.

A good example of a graphical user interface can be found in the ProDom interface. The output from a ProDom query for proteins sharing a homologous domain with a particular sequence may be displayed as 2D graphic representations, summarized alignments and trees, alignment in MSF format, and 3D structures. Specifically, the 2D graphical view presents domain arrangements for proteins sharing homology by showing each protein on a single line, starting with its name, hypertext-linked to SWISS-PROT, followed by a 2D view of schematic boxes, each box hypertext-linked to corresponding ProDom entries.

The limitation of most of these systems is that the graphical displays are both static and unrelated. A static graphical display is defined as when a user is unable to refine or modify the search criteria from within the graphical display. Unrelated graphical displays are defined as when a user modifies a graphical display for a particular search, the remaining graphical displays for the particular search are not correspondingly modified (i.e., no propagation). These limitations can make the analysis of protein sequences cumbersome and time consuming. Additionally, the inflexibility of these system could result in an incorrect or incomplete analysis by limiting a user's ability to view all possible relationships.

Accordingly, there is a need in the art for a user friendly computerized research tool for biological data to provide more effective ways to retrieve and view interrelated information from a database. This system needs to provide a usable display for representing vast amounts of discrete information, permitting researchers to focus on the most relevant materials and discover new functional relationships. To effectually and efficiently analyze the information, there remains a need for a graphical user interface which provides increased flexibility by permitting the user to view any number of related or unrelated data displays from one or more databases at the same time. These displays need to be interlinked such that the selection of one or more entries in one of the display windows causes the other display windows to distinguishably display and act on those entries related to the selection. Progressive querying is also needed to allow the user to quickly discover new relationships based on the results of previous queries.

The present invention is designed to address these needs. --

DISCLOSURE OF THE INVENTION

Broadly speaking, the invention is a computer research tool for searching and displaying biological data. Specifically, the invention provides a computer research tool for performing computerized research of biological data from various databases and for providing a novel graphical user interface that significantly enhances biological data representation, progressive querying and cross-navigation of windows and databases.

The invention can be implemented in numerous ways, including as a system, a device, a method, or a computer readable medium. Several embodiments of the invention are discussed below.

As a computer system, an embodiment of the invention includes a database containing tables of data, a display device and a processor unit. The display device has a plurality of display areas (windows). The processor unit operates to access the database to retrieve the data from the corresponding associated tables and then display the retrieved data in the display areas. The processor unit also detects when a selection associated with one of the display areas is made

and thereafter automatically modifies the data being displayed in the other display areas in accordance with the selection. Selection is made in a graphically distinct manner. Changes in certain selections, including scale and limits, propagate throughout.

As a graphical user interface (GUI) for a display screen of a computer, an embodiment of the invention includes a number of display areas ("windows") for searching and displaying biological data which are interlinked for ease of navigation. A variety of formats for searching and displaying biological data is provided. Searches can be performed by keyword, sequence listing or family identifier (module ID). Sequence search results are graphically displayed showing the relationship between the probe sequence chosen for the query and each of the families that are related with their associated modules. Keyword search results are graphically displayed showing all sequences having the requested keyword along with all modules for the currently selected catalog. The module of interest may then be selected which results in a summary window for the family associated with the module. The family summary window display provides a two dimensional spatial orientation of the biological data, including visual locations of modules (represented as schematic boxes) in each of the sequences for a selected family distinguishably displayed and positionally aligned as well as the location of all other modules in those sequences. Another results display provides the user with the associated multiple sequence alignment of biological data and secondary structure predictions (Vparse, Score, PredSI, PredSec). A further results display provides the user with the associated phylogenetic tree (rooted or unrooted). A further display provides the actual protein sequence information for any selected member of a family.

As a method of displaying data on a display device of a computer system, the data being obtained from a relational database associated with the computer system, the display having "windowing" capability to provide a plurality of display areas, an embodiment of the invention includes the operations of: interlinking display areas via the database such that selections may be propagated throughout. The method further includes multiple catalog views, browsing through families, cross-navigation and propagation through all catalogs and sequences, propagation through families, assigning protein function, and scaling of silent/express mutation ratios.

As a computer readable media containing program instructions for displaying data on a display device of a computer system, the data being obtained from a relational database associated with the computer system, the display having "windowing" capability to provide a plurality of display areas, an embodiment of the invention includes: computer readable code devices for interlinking display areas via the database such that selections may be propagated throughout, multiple catalog views, browsing through families, cross-navigation and propagation

through all catalogs and sequences, propagation through families, assigning protein function, and scaling of silent/express mutation ratios (kA/kS).

The advantages of the invention are numerous. One significant advantage of the invention is that it allows a user to directly use the data returned by one or more queries as the basis for making additional queries. By this kind of interactive and progressive query-making activity, access to all of the information on a given topic is possible. As a result, new data connections and relationships may be discovered. The user is able to more efficiently and effectively review related biological information than conventionally possible.

All patents, patent applications, provisional applications, and publications referred to or cited herein, or from which a claim for benefit of priority has been made, are incorporated herein by reference in their entirety to the extent they are not inconsistent with the explicit teachings of this specification.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is an overview of the preferred embodiment of the hardware architecture for computerized searching of biological data.

FIG. 2 depicts a classic three tier client server model utilized in a preferred embodiment of the present invention (1-tier/Database Server, 2-tier/Application Server, 3-tier/Client).

FIG. 3 depicts the preferred client/server communication model with all three tiers.

FIG. 4A depicts the entity relationship diagram of one embodiment of the database.

FIG. 4B depicts the entity relationship diagram of another embodiment of the database.

FIG. 4C depicts the entity relationship diagram for DNA.

FIG. 5 depicts the block diagram for user related database.

FIG. 6 is a navigational flowchart of a preferred embodiment of the invention illustrating all major windows and options available.

FIG. 7 depicts a sample screen display for the catalog selection window.

FIG. 8 depicts a sample screen display for the search by name window.

FIG. 9 depicts a sample screen display for the Module Family Summary (MFS) window.

FIG. 10 depicts a sample screen display for the search by sequence window.

FIG. 11 depicts a sample screen display for the sequence search results (SSR) window.

FIG. 12 depicts a sample screen display for the search by keyword window.

FIG. 13 depicts a sample screen display for the keyword search results (KSR) window.

FIG. 14 depicts a sample screen display for the MSA window.

FIG. 15 depicts a sample screen display for the evolutionary tree window.

FIG. 16 depicts an example of consecutive screen displays for interactive and progressive query-making activity from search by sequence.

FIG. 17 depicts an example of consecutive screen displays for interactive and progressive query-making activity from search by keyword.

FIG. 18A depicts the MFS window for one catalog.

FIG. 18B depicts the MFS window with a second catalog selected and consecutive screen displays for interactive and progressive query-making activity from this window.

FIG. 19 depicts the screen display for the evolutionary tree window as linked to the MFS window and MSA window with highlighted selections propagated throughout.

FIG. 20 depicts the screen displays showing interactive and progressive query-making activity across multiple MFS windows.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring now to the drawings, the preferred embodiment of the present invention will be described.

I. ARCHITECTURE

Figure 1 is an overview of the preferred embodiment of the hardware architecture for computerized searching of biological data. The architecture preferably comprises at least two networked computer processors (client component and server component(s)) and a database(s) for storing biological data. The computer processors can be processors that are typically found in personal desktop computers (e.g., IBM, Dell, Macintosh), portable computers, mainframes, minicomputers, or other computing devices. Preferably in the networked client/server architecture of the present invention, a classic three tier client server model is utilized as shown in Figure 2 (1-tier/Database Server, 2-tier/Application Server, 3-tier/Client). Preferably, a relational database management system (RDMS), either as part of the Application Server component or as a separate component (RDB machine) provides the interface to the database.

In a preferred database-centric client/server architecture, the client application generally requests data and data-related services from the application server which makes requests to the database server. The server(s) (e.g., either as part of the application server machine or a separate RDB/relational database machine) responds to the client's requests and provides secured access to shared data.

II. CLIENT

More specifically, the client components are preferably complete, stand-alone personal computers offering a full range of power and features to run applications. The client component preferably operates under any operating system and includes communication means, input means, storage means, and display means. The user enters input commands into the computer processor through input means which could comprise a keyboard, mouse, or both. Alternatively, the input means could comprise any device used to transfer information or commands. The display comprises a computer monitor, television, LCD, LED, or any other means to convey information to the user. In a preferred embodiment, the user interface is a graphical user interface (GUI) written and operating under the Java programming language (Sun Microsystems) as a Java compatible browser or Java Virtual Machine (JVM). The GUI provides flexible navigational tools to explore patterns in the evolutionary relationships between genomic sequences. The clients and the Application server communicate via Java's RMI (Remote Method Invocation).

III. SERVER

The server component(s) can be a personal computer, a minicomputer, or a mainframe and offers data management, information sharing between clients, network administration and security. The Database Server (RDBMS - Relational Database Management System) and the Application Server may be the same machine or different hosts if desired. The Application Server is preferably a Java application (JDK Ver. 1.1 or JRE) running on a supported UNIX platform (e.g., Linux, Irix, Solaris). The Database Server is preferably SQL-capable (e.g., MySQL, Oracle). The Application Server and Database Server communicate via the protocol implied by the JDBC (Java Database Connectivity) driver of the RDBMS. The Application Server preferably completely isolates the client from any notion of relational databases; the client's view is one of (Java) objects, not relations.

The present invention also envisions other computing arrangements for the client and server(s), including processing on a single machine such as a mainframe, a collection of machines, or other suitable means.

IV. CLIENT/SERVER COMMUNICATIONS

The client and server machines work together to accomplish the processing of the present invention. The preferable protocol between the client and server is RMI (Remote Method Invocation for Java-to-Java communications across Virtual Machines). RMI is a standard defined by the Java Core.

The isolation of clients from each other requires that each client gets its own server instance as a container of all client related data like database connection or query status. A root

server allows connection bootstrapping by creating server instances. The resulting communication model with all three tiers is depicted in Figure 3.

V. DATABASE HARDWARE

The database is preferably connected to the database server component and can be any device which will hold data. For example, the database can consist of any type of magnetic or optical storing device for a computer (e.g., CDROM, internal hard drive, tape drive). The database can be located remote to the server component (with access via modem or leased line) or locally to the server component.

VI. DATABASE TYPE (RELATIONAL)

The database is preferably a relational database created/derived from existing biological data sets and/or databases (e.g., SwissProt, GeneBank) that is organized and accessed according to relationships between data items. In a preferred embodiment, the database is SQL compatible with standard JDBC supported mechanisms and datatypes. The relational database would preferably consist of a plurality of tables (entities). The rows of a table represent records (collections of information about separate items) and the columns represent fields (particular attributes of a record). In its simplest conception, the relational database is a collection of data entries that "relate" to each other through at least one common field.

In a preferred embodiment, for example, portions of the database may be organized by identifying families of homologous protein sequences within the database, constructing for each family a multiple sequence alignment, an evolutionary tree, and ancestral sequences at nodes in the tree, constructing a corresponding multiple alignment for the DNA sequences that encode the proteins in the protein family, assigning silent and expressed mutations in the DNA sequences to each branch of the DNA evolutionary tree a secondary structure is predicted for the family, and this predicted secondary structure is aligned with the ancestral sequence at the root of the tree.

The predicted structural models and their corresponding models of ancestral sequences may be used to organize the protein sequence database to provide rapid search and retrieval of sequence databases. In this case, to apply the models of secondary structure predicted using the methods disclosed in U.S. Patent No. 5,958,784, incorporated herein by reference, the predicted models are set within the evolutionary history of the protein family. The evolutionary history is defined by a multiple alignment of the sequences of members of the protein family, an evolutionary tree connecting these members, and ancestral sequences reconstructed in probabilistic form throughout the tree.

In the present invention, a multiple alignment, an evolutionary tree, and ancestral sequences at nodes in the tree can be constructed by methods well known in the art for a set of homologous proteins. These three elements of the description are interlocking, as is well known in the art. The presently preferred methods of constructing ancestral sequences for a given tree is the maximum parsimony methods, as implemented (for example) in the commercially available program MacClade [W.P. Maddison, D.R. Maddison, *MacClade, Analysis of Phylogeny and Character Evolution*, Sinauer Associates, Sunderland MA (1992)]. Trees are compared based on their scores using either maximum parsimony or maximum likelihood criteria, and selected based on considerations of score and correspondence to known facts.

Next, a corresponding multiple alignment is constructed by methods well known in the art for the DNA sequences that encode the proteins in the protein family. The multiple alignment is constructed in parallel with the protein alignment. In regions of gaps or ambiguities, the amino acid sequence alignment can be adjusted to give the alignment with the most parsimonious DNA tree. The presently preferred method of constructing ancestral DNA sequences for a given tree is the maximum parsimony method. The DNA and protein trees and multiple alignments must be congruent, meaning that when amino acids are aligned in the protein alignment, the corresponding codons are aligned in the DNA alignment. Likewise, the connectivity of the two evolutionary trees must show the same evolutionary relationships. In regions where the connectivity of the amino acid tree is not uniquely defined by the amino acid sequences, the tree that gives the most parsimonious DNA tree is used to decide between two trees or reconstructions of equal value. Finally, the ancestral amino acids reconstructed at nodes in the tree must correspond to the reconstructed codons at those nodes. When the ancestral sequences are ambiguous, and where the DNA sequences cannot resolve the ambiguity, the reconstructed DNA sequences must be ambiguous in parallel. Approximate reconstructions are valuable even when exact reconstructions are not possible from available data, and the tree is preferably constrained to correspond to evolutionary relationships between proteins inferred from biological data (*e.g.*, cladistics).

Next, mutations in the DNA sequences are then assigned to each branch of the DNA evolutionary tree. These may be fractional mutations to reflect ambiguities in the sequences at the nodes of the tree. When ambiguities are encountered, alternatives are weighted equally. Mutations along each branch are then assigned as being "silent," meaning that they do not have an impact on the encoded protein sequence, and "expressed," meaning that they do have an impact on the encoded protein sequence. Fractional assignments are made in the case of ambiguities in the reconstructed sequences at nodes in a tree.

Thereafter, intermediates in the evolutionary tree are then prepared in the laboratory using protein engineering and biotechnology methods well known in the art [Jermann, T.M., Opitz, J.G., Stackhouse, J., Benner, S. A. "Reconstructing the evolutionary history of the artiodactyl ribonuclease superfamily," *Nature* 374:57-59 (1995)].

The method disclosed in U.S. Patent No. 5,958,784 can then be applied to each protein family. For each protein family, a secondary structure is predicted for the family, and this predicted secondary structure is aligned with the ancestral sequence at the root of the tree. If the root of the tree is unassigned, the predicted secondary structure is aligned with the ancestral sequence calculated for an arbitrary point near the center of gravity of the tree.

As the quality of a multiple alignment and the precision of the reconstructed ancestral sequences decreases if proteins are included in the family with sequences diverging by over 150 PAM units, where a PAM unit is the number of point accepted mutations per 100 amino acids, while the quality of the secondary structure prediction determined by the methods disclosed in U.S. Patent No. 5,958,784 becomes worse if the family does not contain at least some protein sequence pairs 40 PAM units or more divergent, families used in this invention preferably contain at least some protein sequence pairs more than 40 PAM units divergent, but contain no protein pairs more than 150 PAM units divergent. Most preferably, a majority of protein pairs are 40 or more PAM units divergent and no protein pair is more than 120 PAM units divergent. The sequences in a protein family are, however, generally determined by the availability of sequences in the database.

Once the models for secondary structure predicted by the methods disclosed in U.S. Patent No. 5,958,784 are placed into their evolutionary context as described above, the context can be used in the following ways:

1. Rapidly searchable database

The above-noted steps provide one method to organize the protein sequence database in a rapidly searchable form. The ancestral sequences and the predicted secondary structures associated with the families defined by these steps are surrogates for the sequences and structures of the individual proteins that are members of the family. The reconstructed ancestral sequence represents in a single sequence all of the sequences of the descendent proteins. The predicted secondary structure associated with the ancestral sequence represents in a single structural model all of the core secondary structural elements of the descendent proteins. Thus, the ancestral sequences can replace the descendent sequences, and the corresponding core secondary structural models can replace the secondary structures of the descendent proteins.

This makes it possible to define two surrogate databases, one for the sequences, the other for secondary structures. The first surrogate database is the database that collects from each of

the families of proteins in the databases a single ancestral sequence, at the point in the tree that most accurately approximates the root of the tree. If the root cannot be determined, the ancestral sequence chosen for the surrogate sequence database is near the center of mass of the tree. The second surrogate database is a database of the corresponding secondary structural elements. The surrogate databases are much smaller than the complete databases that contain the actual sequences or actual structures for each protein in the family, as each ancestral sequence represents many descendent proteins.

Searching the surrogate databases for homologs of a probe sequence generally proceeds in two steps. First, the probe sequence (or structure) is matched against the database of surrogate sequences (or structures). Should the search yield a significant match, the probe sequence is identified as a member of one of the families already defined. The probe sequence is then matched with the members of this family to determine where it fits within the evolutionary tree defined by the family. The multiple alignment, evolutionary tree, predicted secondary structure and reconstructed ancestral sequences may be different once the new probe sequence is incorporated into the family. If so, the different multiple alignment, evolutionary tree, and predicted secondary structure are recorded, and the modified reconstructed ancestral sequence and structure are incorporated into their respective surrogate databases for future use.

Alignment of ancestral sequences with ancestral sequences has an advantage in detecting longer distance homology, as the ancestral sequences contain information about what amino acid residues are conserved within the nuclear family, and therefore are more likely to be conserved between diverging nuclear families.

VII. DATABASE TABLES

Each separate table (entity) represents a different schema of interconnections between some or all of the protein sequences from the underlying biological data set/ database. Specifically, in a preferred embodiment, the relational database for storing biological data includes a plurality of interrelated tables wherein each table comprises an attribute having a common domain with an attribute of at least one other table in the database. The invention provides for viewing patterns in the evolutionary relationships between genomic sequences on the basis of the data stored in the relational database.

Three versions of this schema can be viewed using the entity relationship diagrams of Figures 4A - 4C which illustrates the fields for each type of data and the interconnections between data types. The following is a detailed description of a collection of tables (entities) in one embodiment of the present invention with the attributes and keys of each relation. The ID field of all relations except those of the SeqAnnType table is assigned automatically when inserting the relation. In particular, Figure 4C shows the entity relationship diagram for DNA.

1. AASequence Table

The AASequence table contains all amino acid sequences available in the database. Every sequence belongs to exactly one sequence database.

#	Name	Type	Key	Description
1	Id	INTEGER	P	The internal id, i.e. the foreign key for all modules, sequence annotations and sequence keys belonging to this sequence.
2	SeqDBId	INTEGER		The sequence database the sequence belongs to.
3	Description	VARCHAR (255)		A description of the sequence.
4	Sequence	LONGVARCHAR		The sequence itself as encoded characters according to Alphabet.getAminoAcidAlphabet ( ).

2. Catalog Table

The Catalog table contains all catalogs available in this database.

#	Name	Type	Key	Description
1	Id	INTEGER	P	The internal id, i.e. the foreign key for all families belonging to this catalog.
2	Name	VARCHAR (64)	S	The unique name for external identification.
3	Description	VARCHAR (255)		A description of the catalog's contents.
4	Version	VARCHAR (16)		The version of this catalog (a date or a version according to some numbering scheme).
5	DBVersion	INTEGER		The database format version of this catalog. The format version is used to verify the compatibility of database and MC server.
6	ProfilePAM	float [ ]		The list of PAM of the available profile databases. It caches the result of a time consuming query.
7	NrModules	INTEGER		The total number of modules in this catalog. It caches the result of a time consuming query.
8	NrFamilies	INTEGER		The total number of families in this catalog. It caches the result of a time consuming query.

3. FamAnnotation Table

The FamAnnotation table contains all annotations of all families. An annotation always belongs to exactly one family.

#	Name	Type	Key	Description
1	Id	INTEGER	P	The internal id.
2	FamId	INTEGER	F	Foreign key to the corresponding family.
3	Annotation	ds. MSAAnnotation		The annotation of the MSA of the family.

4. Family Table

The Family table contains all families of all catalogs. A family always belongs to exactly one catalog and contains an arbitrary number of modules.

#	Name	Type	Key	Description
1	Id	INTEGER	P	The internal id, i.e. the foreign key for all modules, profiles and MSA annotations belonging to this family.
2	Name	VARCHAR (64)	S	The name of this family assigned during modularization.
3	Modification	INTEGER	S	The modification number of this family inside its once assigned name.
4	CatId	INTEGER		Foreign key to the containing catalog.
5	Description	VARCHAR (255)		The description of this family (usually generated automatically during modularization).
6	Tree	ds. Tree		The precalculated phylogenetic tree of this family.

5. Module Table

The Module table contains all modules of all catalogs. A module always belongs to exactly one sequence and exactly one family. The multiple sequence alignment is implicitly stored as a Gaps structure from which the MSA can be directly constructed.

#	Name	Type	Key	Description
1	Id	INTEGER	P	The internal id.
2	SeqId	INTEGER	F	Foreign key to the containing sequence.
3	SeqStart	INTEGER		Start of this module in the containing sequence (o based).
4	SeqLength	INTEGER		Length of this module in the containing sequence.
5	FamId	INTEGER	F	Foreign key to the containing family.
6	FamIndex	INTEGER		Index of this module in the containing family.
7	StartSignificance	REAL		Significance of start point of this module.
8	EndSignificance	REAL		Significance of end point of this module.
9	Quality	REAL		Overall quality of this module.
10	MSAGaps	ds. Gaps		Gaps to be inserted in this module's sequence to obtain the MSA sequence.

6. Profile Table

The Profile table contains all profiles of all families. A profile always belongs to exactly one family. For each family, there may be several profiles at different PAM.

#	Name	Type	Key	Description
1	Id	INTEGER	P	The internal id.
2	FamId	INTEGER	F	Foreign key to corresponding family.
3	Pam	REAL	S	PAM distance from the PAS of the family.
4	Sequence	BLOB		The actual profile sequence in packed representation (byte [ ]).

7. SeqAnnotation Table

The SeqAnnotation table contains all annotations of all sequences in this database. Each annotation belongs to exactly one sequence.

#	Name	Type	Key	Description
1	Id	INTEGER	P	The internal id.
2	SeqId	INTEGER	F	Foreign key to the containing sequence.
3	SeqStart	INTEGER		Start of this annotation in the containing sequence (0 based).
4	SeqLength	INTEGER		Length of this annotation in the containing sequence.
5	Type	INTEGER	F	The sequence annotation type as defined by SeqAnnType.
6	Desscription	VARCHAR (255)		A description of the annotation in addition to the one provided by SeqAnnType. It may be missing, i.e. NULL.

8. SeqAnnType Table

The SeqAnnType table contains all types of all sequence annotations in this database. Its main purpose is to provide standard descriptions for each type to be displayed in the GUI. This entity has more the character of a lookup table than of a real database entity. The Id attribute of each relation is assigned manually. The semantics of some IDs must be known to e.g. the GUI, as different annotations will be displayed differently. Types are collected into groups (type id ranges) whose members normally do not overlap (e.g. secondary structure or binding sites). The groups are predefined in mastercatalog.ds.SeqAnnGroup.

#	Name	Type	Key	Description
1	Id	INTEGER	P	The internal id.
2	Tag	VARCHAR (32)	S	The tag used in the original annotation database corresponding to this type (e.g. a feature table tag).
3	Description	VARCHAR (255)		A description of the annotation.

9. SequenceDB Table

The SequenceDB table contains all sequence databases available in this database.

#	Name	Type	Key	Description
1	Id	INTEGER	P	The internal id, i.e. the foreign key for all sequences belonging to this database and all catalogs built from this database.
2	Name	VARCHAR (64)	S	The unique name for external identification.
3	Description	VARCHAR (255)		A description of the database contents.
4	Version	VARCHAR (16)		The version of this database (a date or a version according to some numbering scheme).
5	SearchKeys	String [ ]		The list of available search keys for sequence searches. It caches the result of a time consuming query.
6	SeqAnnTypes	String [ ]		The list of available sequence annotations. It caches the result of a time consuming query.

10. SequenceKey Table

The SequenceKey table contains all indexed keys of a sequence like ID, accession numbers, EC numbers and so on. Storing these keys separately allows additional key types to be added without modifying the database structure or any code. Furthermore, there may be multiple occurrences of the same key type for any sequence.

#	Name	Type	Key	Description
1	Id	INTEGER	P	The internal id.
2	KeyType	VARCHAR (16)		The key type, e.g. "EC".
3	KeyValue	VARCHAR (255)	S	The key value, e.g. "4.2.1.1".
4	SeqId	INTEGER	F	Foreign key to corresponding sequence.

Short arrays and graphs are preferably stored as BLOBs (Binary Large Objects) to prevent uncontrolled growth of the number of entities in the design. Only large arrays of variable size are stored in their own relation by properly normalizing the database. The mapping between transient Java objects and persistent database relations (rows in a DB table) is based on a unique "Id" (an INTEGER) for each relation. References to other objects in the database are therefore always INTEGER foreign keys.

For each table, there is a subclass whose instances are Java objects corresponding to relations, and a subclass representing the corresponding entity and providing the necessary SQL code.

VIII. GUI

The operation of the present invention will now be described with respect to the graphical user interface and its navigation of the database.

The graphical user interface of the present invention allows the user to browse the sequence database, perform searches, and examine evolutionary relationships. For example, the GUI is a browser that can be used to follow evolutionary relationships through the genomic sequence; the browsing provides interactive trees, multiple sequence alignments, and families; the database of families can be searched using a novel method that represents each family as a probabilistic sequence; rates of evolution are displayed on evolutionary trees and provide evidence of changes in function.

As in most "windows" applications, each display screens of the invention generally comprise a window title bar, a menu bar (with command such as File, Edit, and the like), a tool bar (with options such as Close, Paste, Clear, and the like), and an information display region. The information display region may, for example, display a query window or a results window.

Figure 6 is a navigational flowchart of a preferred embodiment of the invention illustrating all major windows and options available. Rectangles represent the windows and circles represent options available within their respective windows. Arrows indicate direction [and each layer of the program is color codes].

Preferably, the initial window is a LOGIN window (not shown) wherein a user may enter a valid user name and password to access the system.

Upon a successful login, the CATALOG SELECTION WINDOW **100** appears in the user's display as shown in a pictorial diagram in Figure 7 according to an embodiment of the invention. The CATALOG SELECTION WINDOW **100** displays the available catalogs for selection by name, version, description, number of families and number of modules.

Each catalog is constructed to provide a view of relationships between (some or all of) the protein sequences in the database. Different catalogs emphasize different features of the protein sequence database. For example, one catalog might emphasize repeat units within proteins, another catalog focuses on alignments which comprise the whole length of genes (e.g., a gene product catalog), and another focuses on local patterns of divergence between protein sequences (e.g., a modularized catalog). Catalogs are composed of families of modules, each module defining a region of a protein sequence. Thus, the families relate regions of different protein sequences in biologically meaningful ways.

In the database configuration, the Catalog table (described previously) contains the following: Id, Name, Description, Source, Version, DBVersion, ProfilePAM, NrModules, NRFamilies, MinFamId, MaxFamId, SearchKeys, SeqAnnTypes, RefSeqDBs.

Examples of catalogs which may be included in the present invention include the following:

1. Modularized catalog: A description of relationships between proteins sequences based on local patterns of divergence (according to a model of evolution). There are many ways in which such a catalog might be constructed. Published examples include ProDom and DOMO.

2. Gene product catalog: A description between protein sequences one or more of which must comprise whole genes. Example of a method for building a catalog of this type can be seen in Monica Riley's work on Modularization of the E.coli genome.

3. Repeat catalog: A description between regions of proteins which form identifiable repeats of at least 20 residues. Certain features of the DOMO database show features of this type, although this database is more properly defined as a modularized catalog.

4. Entry catalog: A description defining the 'classical' method of sequence database construction. That is that there are no explicit relationships between different sequence entries. It is simply a dictionary of all available protein sequences in the database.

Catalogs can be subsets of all data in the database. Typically this is most (scientifically) useful for the modularized catalogs where focusing on a subset of all the gene products (e.g. just mammalian sequences) is biologically meaningful (e.g., mammalian modularized catalog, bacterial modularized catalog).

Selecting a catalog by double clicking on one of the available catalogs displays a query window for that catalog. Alternatively, you can select a catalog entry with the mouse and click on the Open button on the toolbar.

There is a special catalog, 'the Sequence Entry Catalog' that contains just the individual protein sequences. This catalog necessarily does not have family names or allow sequence searching; it merely provides direct access to individual protein sequences and can be searched by keyword.

Once a catalog is chosen, there are several different ways to search the catalog: a) by catalog family identifier, b) by keyword, and c) by sequence. In a preferred embodiment, the query type is selected using "tabs" associated with each search type. Each search method yields unique results providing a rich mechanism for navigation to related modules. All of these methods may not be available for every catalog (i.e., if there is no family, there would be no family identifier). The methods are described in more detail hereafter.

a). Figure 8 - Search by Name (Family identifier/Module ID): Each family in a catalog preferably has a unique identifier that defines a particular family. Preferably, these family names are generated automatically when each catalog is built from the protein sequence data. This search window **110** allows you to obtain a specific family number of interest as shown in Figure 8. The result of this search is the Module Family Summary (MFS)window **140** of Figure 9 showing a graphical view of the associated proteins and their modules.

b). Figure 10 - Search by Sequence: This window **120** allows you to search a protein against a catalog for homologous (evolutionarily related) protein families as shown in Figure 10. Homology is an important concept in extracting information from sequence databases because conclusions can be drawn about the chemical behavior and biological function when two proteins are homologous. One way to determine whether two proteins are homologous is to compare their amino acid sequences. Procedures are well established in the art for comparing two protein sequences, scoring similarities, and using this score to assess the likelihood that the similarities arose by reason of common ancestry rather than by random chance [Gonnet, G.H., Cohen, M.A., Benner, S.A. Exhaustive matching of the entire protein sequence database. *Science* 256, 1443-1445 (1992)]. You can enter the amino acid of interest, specify a minimum scope and maximum amount of matches, adjust the PAM sensitivity (the sensitivity of the search cen be set, for example, at 50 PAM to look for close homologs or 150 PAM to detect long distance

homology. Only the hits above the specified minimum score and up to the maximum amount of matches are reported.). This search can take a while for large probe sequences. You can abort the search at any time with the cancel button.

The result of this search is a Sequence Search Results (SSR) window **125** showing a graphical view of the relationship between the probe sequence and the associated families as shown in Figure 11. This window shows the relationship between the probe sequence chosen for the query and each of the families that are related. The score corresponds to a 'log odds score', the probability that the relationship between the sequence is related according to the model of evolution vs. the probability that the similarity is by chance.

Double clicking on any module shown in SSR window will display the module family summary (MFS) window **140** for that family. The alignment of the probe sequence with the summary can then be displayed.

c). Figure 12 - Search by Keyword: This window **130** provides searches of the protein sequence database by keyword, according to annotations of proteins in the original sequence database as shown in Figure 12. Keywords provided include selection by organism, by classification, by gene name and by gene product description.

The result of this search is a keyword search results (KSR) window **135**, Figure 13, includes a list of database sequence ID's which have database sequence annotations matching the keywords in the description. The display shows a graphical view of the individual protein sequences which fit the keyword search criteria. The graphical view is shown as a linear arrangement of schematic boxes (spatially oriented) representing the existing modules found in the selected catalog along the identified amino acid range (AA). Each of the schematic boxes is preferably identified with its corresponding Module ID number and is differentiated by color or another form of distinctive representation. In some cases, sequences may match with a particular keyword but no modularization information is available (and no schematic boxes shown) because these sequences were not part of the set included in the currently selected catalog.

Double clicking on any module shown in the KSR window **135** will display the module family summary (MFS) window **140** for the associated family.

The Module Family Summary (MFS)window **140** of Figure 9 showing a graphical view of the associated proteins and their modules will now be described in detail . This window is the gateway to navigational power of the present invention by providing a gateway to other displays. The window shows all of the sequences in the currently selected catalog that are members of a particular family, where the family contains all of the sequences which have a particular module of interest. All relationships between modules have been precalculated in the database. The

module is a subsequence that is a member of family where the graphical length is proportionate. Unidentified regions do not have schematic boxes. The module of interest is preferably visibly distinguished from the other modules and its ID is identified in the title bar of the window. The sequences in the family may also contain other modules. The sequences are ordered to cluster modules which are closest together in evolutionary distance.

The window is preferably tabular with a separate numbered row for each sequence. The columns preferably include the sequence ID, a description, the amino acid range and a graphical view of the sequence shown as a linear arrangement of schematic boxes (spatially oriented) representing the existing modules found in the selected catalog along the identified amino acid range (AA). Each of the schematic boxes is preferably identified with its corresponding Module ID number and is differentiated by color or another form of distinctive representation. The schematic boxes for the currently selected module are vertically aligned and visually distinguished, such as by color (red). The windows "tool tips" feature may expand any truncated descriptions or provide additional information in a floating window when the pointer is placed over a particular table entry. This window provides a good indication of any long distance homology between various modules. Proteins that share a common module frequently possess other homologous modules at analogous positions. Such relationships can be confirmed by examining multiple sequence alignments and trees.

The toolbar of the MFS window allows you to perform many different tasks from this point (e.g., print, export to disk, display multiple sequence alignment (MSA), and display phylogenetic tree). Each of these tasks will now be discussed in detail. The family summary can be printed directly to a printer available from a local computer. The description of the modularization of this family can be exported to file.

We now turn to Figure 14 for the display the multiple sequence alignment (MSA) of the current family. Selecting the MSA button on the window's toolbar shows the multiple sequence alignment (the way in which the modules are related at the amino acid level) in the MSA window **150** of Figure 14. This window provides detailed evolutionary information at the protein sequence level to following pattern of conservation and variation of amino acid composition of the module. The MSA is preferably colored according to hydrophobic or hydrophilic nature of the amino acids. (e.g., RED indicates hydrophobic propensity and BLUE indicates hydrophillic). The numbering system used in the MSA window preferably corresponds to the number system implemented by both the MFS window and the Tree window. Highlighted on each sequence are annotated regions of individual protein sequences. Moving the cursor (pointer) over a chosen highlighted region displays the annotation in a floating window. These can be hand-crafted comments, such as feature table entries from SwissProt or automatically generated from patterns

such as those in PROSITE (or others such as PRINTS). Different annotations can be selected from the option bar at the bottom of the MSA window.

Analyzing correlations in the patterns of substitution in the sequences for each module family allows predictions to be made about the nature of underlying structural or functional constraints. Preferably, the annotations provides are VParse, showing the location of putative structure breaking residues or motifs; Score, showing the degree of conservation at each position in the alignment. This value is dependent on both the evolutionary distances between the sequences and the mutability of the individual amino acids, and is a sensitive indicator of significance of conserved sites; PredSI, indicating the predicted solvent accessibility of the residue at that position; and PredSec, indicating the predicted secondary structure. If the PAM width of the family is poor or the number of sequences is small, then there may be insufficient information for a secondary structure prediction. If the given module aligns significantly with any entry in the PDB indicating a confidant homology, a string of secondary structural elements corresponding to that alignment can be seen at the bottom of the MSA window below PredSI and PredSec strings.

As previously described, the multiple sequence alignment window shows the amino-acid by amino-acid relationship between proteins which are in the same family. Some preferred features include a) Coloring: Hydrophobic residues in red, hydrophilic residues in blue, amphiphilic residues in black, b) Parses: Regions of sequence which are likely to represent secondary structure breaking positions, are indicated, c) PredSI: Predicted surface/interior residues are indicated, d) PredSec: Predicted secondary structure is shown, e) Experimental Sec: If known, the secondary structure of experimentally determined homologs to the family are shown, f)Annotation: Sequence features, such as motifs of well known function can be selected from the 'annotations:' list box in the window (see below). Functions of the MSA window include: a) Export sequence data: The sequence of selected modules of this family can be exported to file in a variety of formats, b)Copy sequence data to clipboard: The sequence of selected modules of this family is copied to the clipboard when this button is selected, c) Print: The family summary can be printed directly to any printer available from your local computer, d) Annotations: The sequences in the MSA can be highlighted for particular annotations (usually specific sequence motifs or special database annotations). One such collection of annotations is the 'Prosite' database. Regions of sequence corresponding to Prosite annotations are colored in orange. Moving the cursor over that region of the text displays the details of the annotation in a floating window. The complete set of annotations visible in the current window can be obtained by looking at the 'Annotation Types' menu. A subset of all the annotations in the current collection can be obtained by customizing with the tickboxes in the annotation menu.

Selection of sequences/modules will now be described. You can select some or all of the sequences of individual modules. Sequences are selected individually in the MSA display with single mouse clicks. To combine your selection with previous selections, keep the <CTRL> key depressed while selecting. In addition to the toolbar tasks, other features are available from the window. For example, double clicking on any sequence in the ID column shows the protein sequence window for that family (including catalog membership, description, and annotations).

We now turn to Figure 15 for the display the evolutionary tree of the current family. Selecting the Tree button on the window's shows the evolutionary tree in the Tree window **160** of Figure 15. This indicates the pattern of divergence/similarity between individual modules, assuming that the distance between modules can be computed from the similarity in the protein sequences. Specifically, trees show an estimate of the evolutionary history of a protein module, constructed using the PAM distances between individual members of that family. Trees may be displayed either as rooted or unrooted form; there is no significant distinction between these representations, the location of the root being chosen to balance the tree.

On the branches of the tree, the length of the branches are displayed in PAM units. This provides an estimate of divergence in composition of the various sequences. Selecting the "kA/kS" key at the bottom of the window will display the ratio of the rate of expressed changes at the DNA level to the rate of silent changes, i.e., the rate of mutation leading to changes at the protein level calibrated against the rate of mutation leading to changes only at the DNA level.

The rates are preferably normalized so that when reading expressed:silent ratios, a value of around 1.0 indicates no selection, both synonymous (silent changes) and non-synonymous (expressed changes) substitutions being equally likely (e.g., a pseudogene). The threshold level of kA/kS is user adjustable on the slide scale. Separate coloring schemes are preferably used to indicate branches above or below the threshold. For example, if kA/kS is less than the threshold (default 1.0) the branch is colored blue and if kA/ kS is greater than the threshold, the branch is colored red. If no DNA sequence information is available, then the branch is colored black. In practice, proteins are normally under the influence of purifying selection so the ratios fall well below this value. Therefore, where the ratio approaches or exceeds 1.0, the confidence that one is looking at an episode of rapid sequence evolution (presumably to new function) increases. For adaptations which occurred at longer evolutionary times, the value of the expressed:silent ratio will appear lower as random mutation has increased the number of silent changes. A suitable threshold value can be determined by examining the tree as a whole, which will contain branches that have maintained purifying selection for longer periods and comparing these values with those that suggest mare rapid changes at expressed sites.

The graphic interface also offers scaling facility to zoom in and zoom out using the slide scale at the bottom of the window. Zooming may be necessary in order to see the PAM distance labels or leaf identifiers. This information can also be seen in a floating window by positioning the cursor over the leaves or branches. Individual branches can be displayed on separate trees by selecting the appropriate branches and the "Zoom" key. The "fit" button sizes the tree to the full window size.

As previously described, the Tree window shows the evolutionary relationship between individual modules of a family, using distance calculated from a comparison of their amino acid sequences. Some preferred features include: a) Coloring: Blue edge - KaKs (see below) below threshold, Red edge - KaKs above threshold, Black edge - KaKs not computed (no DNA/unreliable DNA sequence). The threshold is selected by changing a slide bar on the toolbar of the Tree window, b) Rooted/Unrooted: The tree can be displayed in rooted, or in unrooted form, depending on user preference. There is no difference in information content between these two descriptions; the root of the tree is chosen for balance, not as a result of other phylogenetic evidence. Some preferred function include: a) Export tree description: The tree can be exported to file in a variety of formats, b) Print: The tree can be printed directly to any printer available from your local computer, c) Annotations: The sequences in the MSA can be highlighted for particular annotations (usually specific sequence motifs or special database annotations). One such collection of annotations is the 'Prosite' database. Regions of sequence corresponding to Prosite annotations are colored in orange. Moving the cursor over that region of the text displays the details of the annotation in a floating window. The complete set of annotations visible in the current window can be obtained by looking at the 'Annotation Types' menu. A subset of all the annotations in the current collection can be obtained by customizing with the tickboxes in the annotation menu; d) Selection sequences/modules: You can select branches or leaves of the tree with single clicks of the mouse. Selecting a branch will result in all of the branches and leaves being selected downstream of the root. (The root is marked with a circle). To combine your selection with previous selections, keep the <CTRL> key depressed while selecting; e) Fit tree/Rescale tree: The fit button can be used to rescale the tree to fit into the entire window. Pressing SHIFT and the left mouse button can be used to zoom in toward the selected portion of the window, and SHIFT with the right mouse button will zoom out from the selected portion of the window; f) Ka/Ks: You can display either KaKs or PAM distances on the tree.

The MFS window, tree window and MSA window are linked so that selections in one window highlight sequences in the other. You can select some or all of the modules possessed by individual sequences. Modules are selected individually in the MFS window with single

mouse clicks. All of the modules possessed by a protein can be selected at once by selecting the module id (#) on the left hand column. To combine your selection with previous selections, keep the <CTRL> key depressed while selecting.

Normally selections propagate through displays of the current family, i.e., selecting sequences in any window that applies to a family highlights that module in every display for that family - the Tree window, the MSA window and the MFS window. If the propagate check box is set, selections propagate by sequence - i.e., all proteins sequences possessing any selected modules are highlighted. This mode is extremely useful when tracing relationships across different protein families.

We will now describe navigation to another family (related by membership with sequences in the current family). Double clicking on any module in the graphical representation displays the family summary window corresponding to that module.

Notable features of the present invention include a) multi-catalog views where a user can simultaneously view more then one catalog, b) tree to tree interactivity where active selections from a current window get propagated throughout (selected and deleted), c) connectivity of selections where a section of a tree as selected will highlight associated information in other windows which is continuously applied as windows are opened, d) the MSA window's Prosite annotations, tool tips to show annotations, and customization of annotation display/view to select subsets.

Appendix A [20 pages] contains source code for certain of the functions of the present invention written in Java, including IndexSelection.java for managing and propagating selections, FamilyFrame.java for showing the data about a family as a table, FamilyTableModel.java for representation of family information with sequences and their modules, IndexSelectionListener.java for as an interface for a class which listens to index selection changes, and FamilySequenceRenderer.java for rendering family sequences wherein single clicking on a module selects it and double clicking triggers opening of the corresponding family frame.

Following are examples which illustrate procedures for practicing the invention. These examples should not be construed as limiting.

Example 1 –

FIG. 16 depicts an example of consecutive screen displays for interactive and progressive query-making activity from search by sequence. First a sequence is typed or cut and pasted into the query box **120**. Minimum scope, maximum matches and PAM are adjusted by

the user as desired. The query is then run resulting in the Sequence Search Results (SSR) window **125**. From this window, the user can progressively query the desired module(s) by double-clicking on that module(s) which results in the MFS window **140** with the module of interest designated in RED and positionally aligned with all other modules.

Example 2 --

FIG. 17 depicts an example of consecutive screen displays for interactive and progressive query-making activity from search by keyword. The search term "isocitrate" is entered (other search terms may be included with the necessary boolean logic) in the query window **130**. After the query is run, the Keyword Search Results (KSR) window **135** is displayed listing all sequences which contain the queried keyword with the graphical display of the sequences and their modules. From this window, the user may select a module of interest to be displayed in the MFS window **140**. The MFS window **140** shows the two-dimensional spatial orientation of the biological data, including visual locations of modules (represented as schematic boxes) in each of the sequences for a selected family distinguishably displayed and positionally aligned as well as the location of all other modules in those sequences.

Example 3 --

FIG. 18A depicts the MFS window **140** for one catalog. From the menu bar, additional catalog views may be selected. In this example, both the genomes and OPgenomes catalogs are chosen to be viewed as shown in FIG. 18B. This view depicts the MFS window with two catalogs. From this window the user can begin progressive query-making activity by selecting modules of interest and viewing them in the MFS windows.

Example 4 --

FIG. 19 depicts the screen display for the evolutionary tree window **160** as linked to the MFS window **140** and MSA window **150** with highlighted selections propagated throughout. Highlighting a section of the tree will automatically propagate the highlighting to the other windows (MSA and MFS) for the selected sequences.

Example 5 --

FIG. 20 depicts the screen displays showing interactive and progressive query-making activity across multiple MFS windows **140**. Beginning, for example, with the 358_1 module MFS window, the user can select the 377_1 for display, the 978_1 for display, and the 371_1 for display (simultaneously). The windows can be closed or moved about the screen as desired. Thereafter, the user can continue to select displays from the resulting windows, *e.g.*, the user can select the 1075_1 module from the 978_1 MFS window for display, and so on. Progressive querying can be continued through level upon level. Of course, from each MFS window **140**, the

MSA **150** of Figure 14, evolutionary tree **160** of Figure 15, or database entry (not shown) can be displayed as depicted in the flowchart of Figure 6.

It should be understood that the examples and embodiments described herein are for illustrative purposes only and that various modifications or changes in light thereof will be suggested to persons skilled in the art and are to be included within the spirit and purview of this application and the scope of the appended claims. All patents, patent applications, provisional applications, and publications referred to or cited herein, or for which a claim for benefit of priority has been made, are incorporated by reference in their entirety to the extent they are not inconsistent with the explicit teachings of this specification.

CLAIMS

What is claimed is:

1. A method of searching and displaying biological data such that patterns in the evolutionary relationships between genomic sequences can be explored, comprising the following steps:

(a) selecting at least one catalog, wherein the catalog comprises an organized body of related biological data;

(b) searching the catalog using a probe sequence to obtain a listing of search results displayed in graphical form which shows the relationship between the probe sequence and each of the modules that are evolutionarily related to the probe sequence, wherein a module is a region of a protein sequence;

(c) selecting a module of interest from the search results listing; and

(d) displaying a family which comprises a set of all sequences having the selected module of interest wherein each sequence of the set includes a corresponding graphical representation of the various modules of the sequence along its amino acid range.

2. The method of claim 1 further comprising the step of displaying a multiple sequence alignment, evolutionary tree or database entry for the family.

3. The method of claim 1 wherein the graphical representation comprises a two-dimensional spatially oriented view for each sequence of the set wherein the modules for each sequence are represented as schematic boxes and wherein the modules for each sequence are sequentially ordered along its amino acid range.

4. The method of claim 3 wherein like modules for each sequence of the family are positionally aligned with other like modules and visibly distinguished.

5. The method of claim 1 further comprising the step of selecting a new module of interest from the graphical representation of the various modules and displaying a different family which comprises a set of all sequences having the new selected module of interest.

6. The method of claim 5 wherein each sequence of the set includes a corresponding graphical representation of the various modules of the sequence along its amino acid range.

7. The method of claim 1 further comprising the step of adding at least one more catalog so that the graphical display of the family includes a corresponding graphical representation of the various modules of the sequence for each catalog.

8. The method of claim 7 further comprising the step of selecting a new module of interest from the graphical representation of the various modules from any of the catalogs and displaying, for that catalog, a different family which comprises a set of all sequences having the new selected module of interest.

9. The method of claim 2 wherein in the multiple sequence alignment shows the amino-acid by amino-acid relationship between proteins which are in the same family.

10. The method of claim 9 wherein the hydrophobic, hydrophilic, and amphiphilic residues are visually distinguished.

11. The method of claim 9 wherein the regions of sequence which are likely to represent secondary structure breaking positions are visually indicated.

12. The method of claim 9 wherein the predicated surface/interior residues are visually indicated.

13. The method of claim 9 wherein the predicted secondary structure is shown.

14. The method of claim 9 wherein the secondary structure of experimentally determined homologs to the family are shown if known.

15. The method of 2 wherein the evolutionary tree is displayed to indicate the pattern of divergence and similarity between individual modules .

16. The method of claim 15 wherein the evolutionary tree is constructed using PAM distances between individual members of the family.

17. The method of claim 16 wherein the ratio of expressed changes at the DNA level to the rate of silent changes is displayed.

18. The method of claim 17 wherein separate coloring schemes are used to indicate branches above or below a user selectable threshold for the ratio.

19. The method of claim 2 wherein the display of the graphical representation of the various modules of the sequence of a family, the display of the multiple sequence alignment, and the display of the evolutionary tree are related such that a user selection in any of these displays is propagated through the other displays.

20. A method of searching and displaying biological data such that patterns in the evolutionary relationships between genomic sequences can be explored, comprising the following steps:

(a) selecting at least one catalog, wherein the catalog comprises an organized body of related biological data;

(b) searching the catalog by keyword to obtain a listing of search results displayed in graphical form which shows individual protein sequences which match the keyword description, wherein specific regions of each protein sequence are visibly distinguished as modules;

(c) selecting a module of interest from the search results listing; and

(d) displaying a family which comprises a set of all sequences having the selected module of interest wherein each sequence of the set includes a corresponding graphical representation of the various modules of the sequence along its amino acid range.

21. A computer system for searching and displaying biological data such that patterns in the evolutionary relationships between genomic sequences can be explored, comprising:

input means for selecting at least one catalog, wherein the catalog comprises an organized body of related biological data;

processing means for searching the catalog using a probe sequence to obtain a listing of search results displayed in graphical form which shows the relationship between the probe sequence and each of the modules that are evolutionarily related to the probe sequence, wherein a module is a region of a protein sequence;

input means for selecting a module of interest from the search results listing; and

display means for displaying a family which comprises a set of all sequences having the selected module of interest wherein each sequence of the set includes a corresponding graphical representation of the various modules of the sequence along its amino acid range.

22. A graphical user interface for searching and displaying biological data such that patterns in the evolutionary relationships between genomic sequences can be explored, comprising:

a first display area for selecting at least one catalog, wherein the catalog comprises an organized body of related biological data;

a second display area for searching the catalog using a probe sequence;

a third display area which provides a listing of search results displayed in graphical form which shows the relationship between the probe sequence and each of the modules that are evolutionarily related to the probe sequence, wherein a module is a region of a protein sequence, and wherein a module of interest from the search results listing can be selected; and

a fourth display area for displaying a family which comprises a set of all sequences having the selected module of interest wherein each sequence of the set includes a corresponding graphical representation of the various modules of the sequence along its amino acid range.

23. A graphical user interface for displaying biological data such that patterns in the evolutionary relationships between genomic sequences can be explored, comprising:

a display area for displaying a family which comprises a set of all sequences having a selected module of interest wherein each sequence of the set includes a corresponding graphical representation of the various modules of the sequence along its amino acid range and wherein the graphical representation comprises a two-dimensional spatially oriented view for each sequence of the set wherein the modules for each sequence are represented as schematic boxes and wherein the modules for each sequence are sequentially ordered along its amino acid range.

24. The graphical user interface of claim 23 wherein like modules for each sequence of the family are positionally aligned with other like modules and visibly distinguished.

25. A computer readable media containing program instructions for displaying data on a display device of a computer system, the data being obtained from tables in a database associated with the computer system, said computer readable media comprising:

first computer program code for selecting at least one catalog, wherein the catalog comprises an organized body of related biological data;

second computer program code for searching the catalog using a probe sequence to obtain a listing of search results displayed in graphical form which shows the relationship between the probe sequence and each of the modules that are evolutionarily related to the probe sequence, wherein a module is a region of a protein sequence;

third computer program code for selecting a module of interest from the search results listing; and

fourth computer program code for displaying a family which comprises a set of all sequences having the selected module of interest wherein each sequence of the set includes a corresponding graphical representation of the various modules of the sequence along its amino acid range.

26. A computerized storage and retrieval system of biological information comprising a data storage means for storing data in a relational database wherein the database comprises tables, each table having a domain of at least one attribute in common with at least one other table, said tables comprising:

at least one table for storing all amino acid sequences available in the database;

at least one table for storing all catalogs available in the database;

at least one table for storing all annotations of all families;

at least one table for storing all families of all catalogs;

at least one table for storing all modules of all catalogs;

at least one table for storing all profiles of all families;

at least one table for storing all annotations of all sequences in the database;

at least one table for storing all types of sequence annotations in the database;

at least one table for storing all sequence databases available in the database; and

at least one table for storing all indexed keys of a sequence.

27. A computer system for storing and retrieving biological data comprising:

a relational database for storing biological data comprising a plurality of interrelated tables wherein each table comprises an attribute having a common domain with an attribute of at least one other table in the database; and

means for viewing patterns in the evolutionary relationships between genomic sequences on the basis of the data stored in the relational database.

28. The computer system of claim 27 wherein the database comprises tables, wherein the database comprises tables, said tables comprising:

at least one table for storing all amino acid sequences available in the database;

at least one table for storing all catalogs available in the database;

at least one table for storing all annotations of all families;

at least one table for storing all families of all catalogs;

at least one table for storing all modules of all catalogs;
at least one table for storing all profiles of all families;
at least one table for storing all annotations of all sequences in the database;
at least one table for storing all types of sequence annotations in the database;
at least one table for storing all sequence databases available in the database; and
at least one table for storing all indexed keys of a sequence.

29. A computer system for storing and retrieving biological data comprising:
a database comprising tables wherein said biological information is stored such that the tables are interrelated by having at least one common attribute;
means for viewing patterns in the evolutionary relationships between genomic sequences on the basis of the data stored in the database.

30. A method of graphically representing on a display device information about long distance homology between numerous modules, each specific module comprising a common subsequence, the method comprising the steps of:
selecting a module of interest; and
displaying a set of all proteins in a database possessing said module of interest, each protein in the set having a graphical view of its modules wherein the selected module of interest and any other homologous modules at analogous positions are visually distinguished.

31. In a database organized by identifying families of homologous protein sequences within the database, constructing for each family a multiple sequence alignment, an evolutionary tree, and ancestral sequences at nodes in the tree, constructing a corresponding multiple alignment for the DNA sequences that encode the proteins in the protein family, assigning silent and expressed mutations in the DNA sequences to each branch of the DNA evolutionary tree, a method of graphically representing on a display device information about long distance homology between numerous modules, each specific module comprising a common subsequence, the method comprising the steps of:
selecting a module of interest; and
displaying a set of all proteins in a database possessing said module of interest, each protein in the set having a graphical view of its modules wherein the selected module of interest and any other homologous modules at analogous positions are visually distinguished.

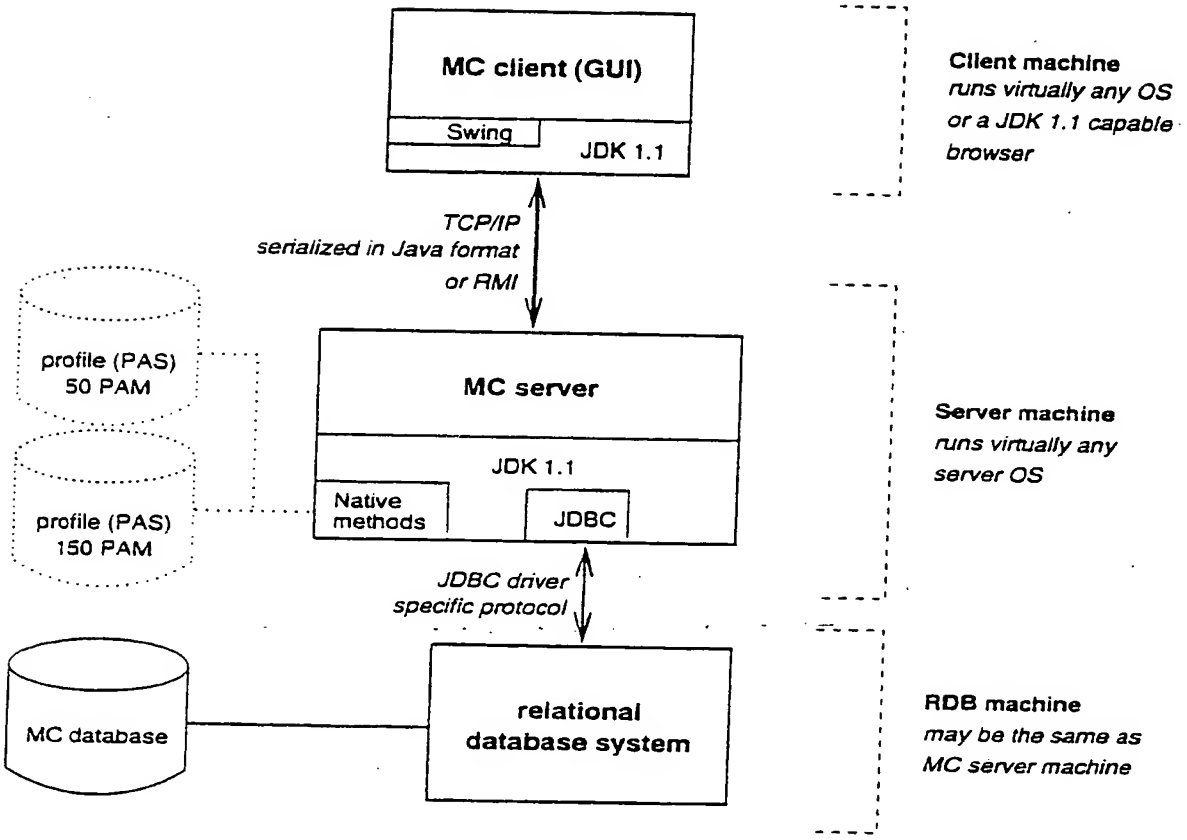

MC client (GUI)
Swing
JDK 1.1
Client machine
runs virtually any OS
or a JDK 1.1 capable
browser
TCP/IP
serialized in Java format
or RMI
profile (PAS)
50 PAM
profile (PAS)
150 PAM
MC server
JDK 1.1
Native
methods
JDBC
Server machine
runs virtually any
server OS
JDBC driver
specific protocol
MC database
relational
database system
RDB machine
may be the same as
MC server machine

FIGURE 1

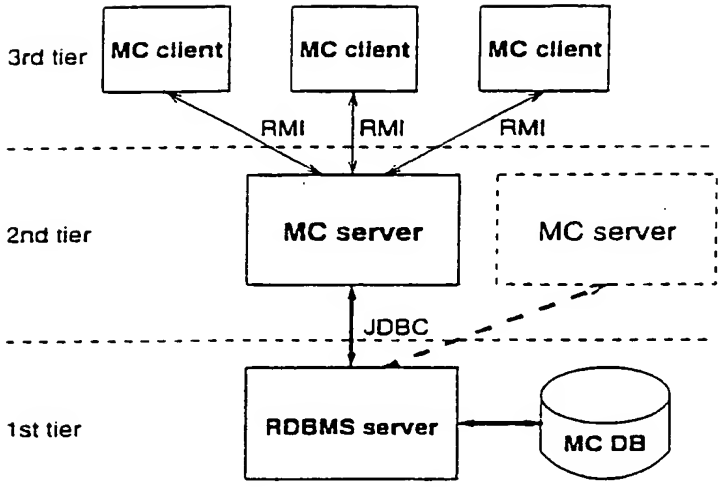

3rd tier
MC client
MC client
MC client
RMI
RMI
RMI
2nd tier
MC server
MC server
JDBC
1st tier
RDBMS server
MC DB

FIGURE 2

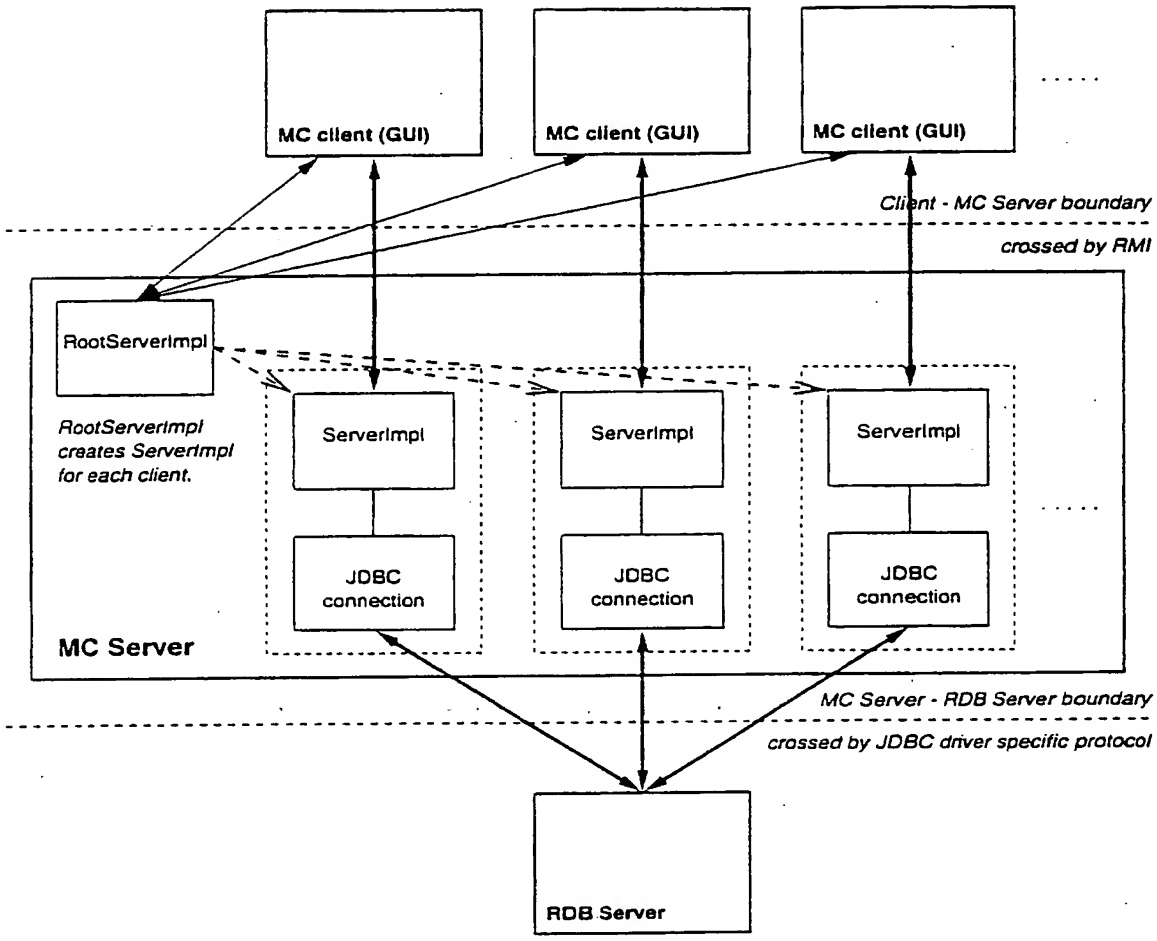

MC client (GUI)
MC client (GUI)
MC client (GUI)
Client - MC Server boundary
crossed by RMI
RootServerImpl
RootServerImpl creates ServerImpl for each client.
ServerImpl
ServerImpl
ServerImpl
JDBC connection
JDBC connection
JDBC connection
MC Server
MC Server - RDB Server boundary
crossed by JDBC driver specific protocol
RDB Server

FIGURE 3

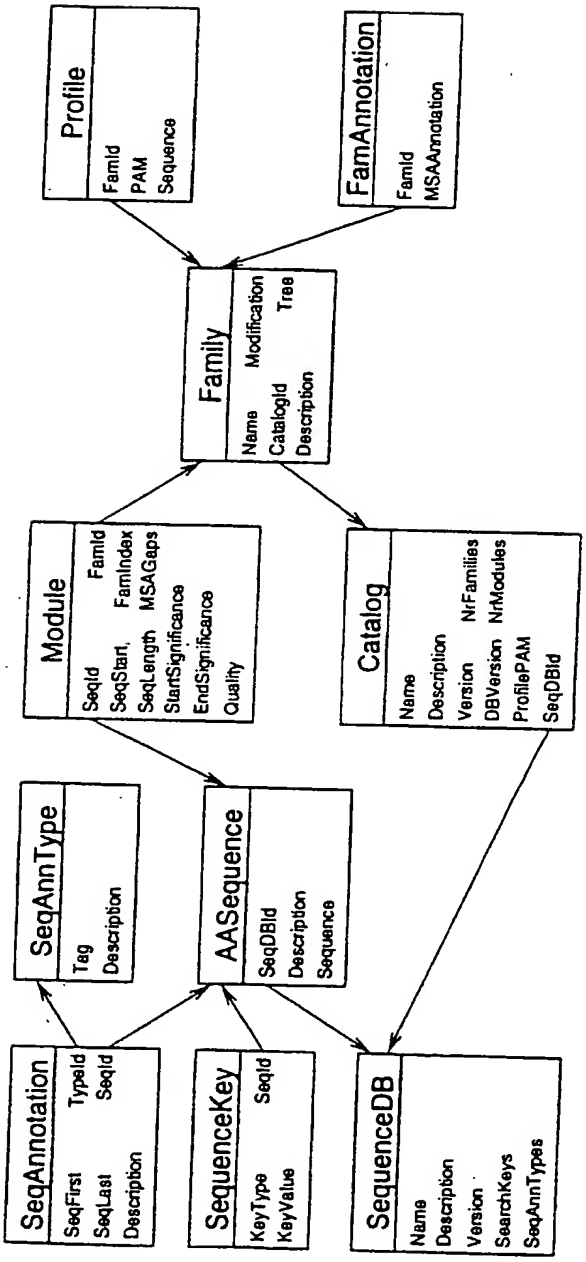

Profile
FamId
PAM
Sequence
FamAnnotation
FamId
MSAAnnotation
Family
Name
CatalogId
Description
Modification
Tree
Module
SeqId
SeqStart
SeqLength
StartSignificance
EndSignificance
Quality
FamId
FamIndex
MSAGaps
Catalog
Name
Description
Version
DBVersion
ProfilePAM
SeqDBId
NrFamilies
NrModules
SeqAnnType
Tag
Description
AASequence
SeqDBId
Description
Sequence
SeqAnnotation
SeqFirst
SeqLast
Description
TypeId
SeqId
SequenceKey
KeyType
KeyValue
SeqId
SequenceDB
Name
Description
Version
SearchKeys
SeqAnnTypes

FIGURE 4A

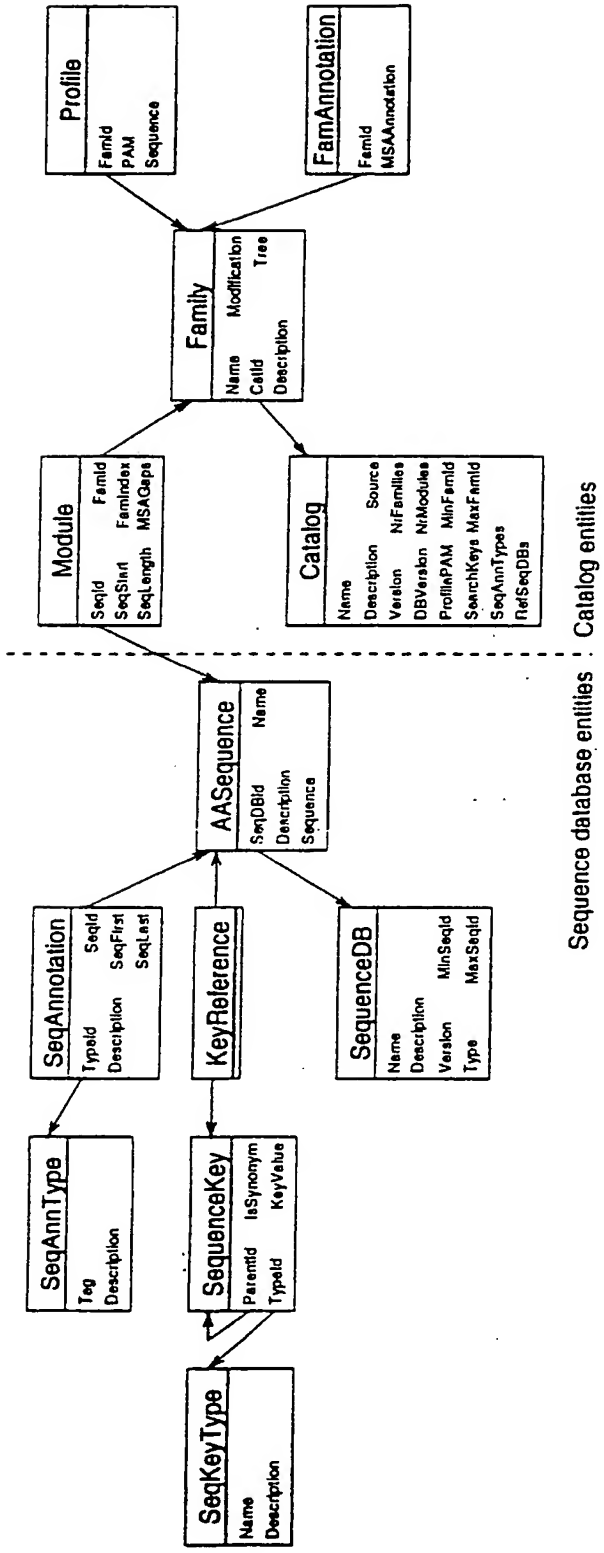

Profile
FamId
PAM
Sequence
FamAnnotation
FamId
MSAAnnotation
Family
Name
CatId
Description
Modification
Tree
Module
SeqId
SeqStart
SeqLength
FamId
FamIndex
MSAGaps
Catalog
Name
Description
Version
DBVersion
ProfilePAM
SearchKeys
SeqAnnTypes
RefSeqDBs
Source
NrFamilies
NrModules
MinFamId
MaxFamId
Catalog entities
Sequence database entities
AASequence
SeqDBId
Description
Sequence
Name
SeqAnnotation
TypeId
Description
SeqId
SeqFirst
SeqLast
KeyReference
SequenceDB
Name
Description
Version
Type
MinSeqId
MaxSeqId
SeqAnnType
Tag
Description
SequenceKey
ParentId
TypeId
IsSynonym
KeyValue
SeqKeyType
Name
Description

FIGURE 4B

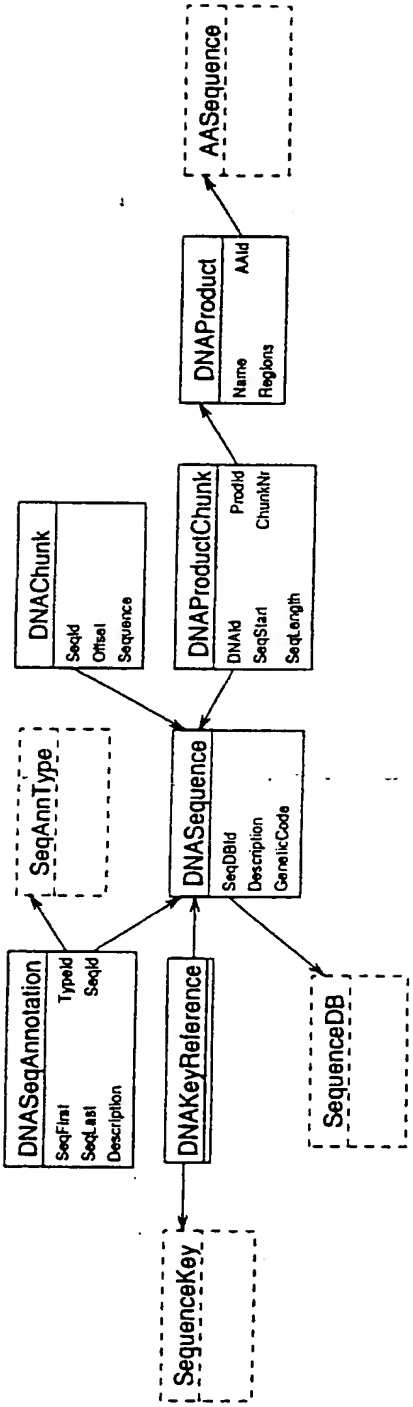

AASequence
DNAProduct
AAId
Name
Regions
DNAProductChunk
ProdId
ChunkNr
DNAId
SeqStart
SeqLength
DNAChunk
SeqId
Offset
Sequence
SeqAnnType
DNASequence
SeqDBId
Description
GeneticCode
DNASeqAnnotation
TypeId
SeqId
SeqFirst
SeqLast
Description
DNAKeyReference
SequenceDB
SequenceKey

FIGURE 4C

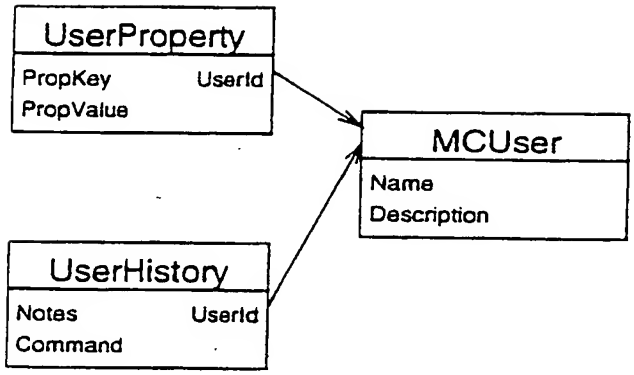

UserProperty
PropKey
UserId
PropValue
MCUser
Name
Description
UserHistory
Notes
UserId
Command

FIGURE 5

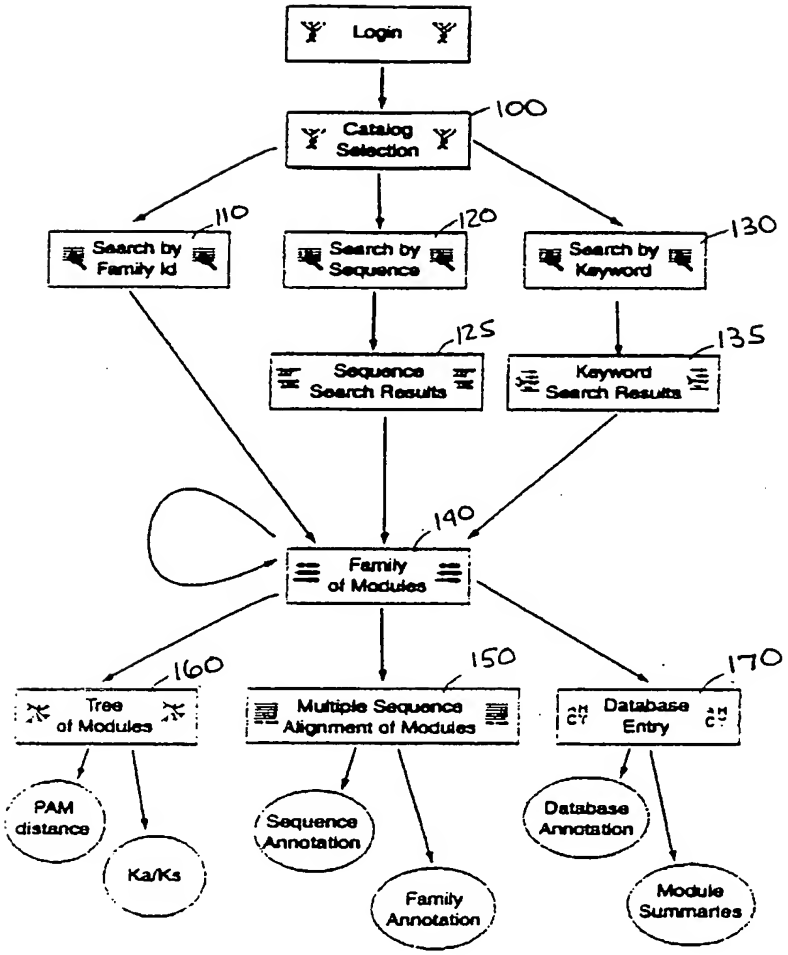
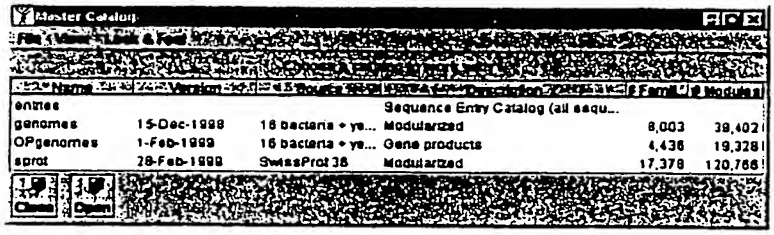

Login
Catalog Selection
100
Search by Family Id
110
Search by Sequence
120
Search by Keyword
130
Sequence Search Results
125
Keyword Search Results
135
Family of Modules
140
Tree of Modules
160
Multiple Sequence Alignment of Modules
150
Database Entry
170
PAM distance
Ka/Ks
Sequence Annotation
Family Annotation
Database Annotation
Module Summaries

FIGURE 6



100

FIGURE 7

Catalog Window

This window lists the available catalogs.

Each catalog is as a view of relationships between (some or all of) the protein sequences in the database. Different catalogs emphasize different features of the protein sequence database. (For example, one catalog might emphasize repeat units within proteins, another catalog focuses on alignments which comprise the whole length of genes (the gene product catalog), and another focuses on local patterns of divergence between protein sequences (the modularized catalog). Catalogs are composed of families of modules, each module defining a region of a protein sequence. Thus, the families relate regions of different protein sequences in biologically meaningful ways.

Double clicking on one of the available catalogs displays a query window for that catalog. Alternatively, you can select a catalog entry with the mouse and click on the Open button on the toolbar.

Note: There is a special catalog, 'the Sequence Entry Catalog' that contains just the individual protein sequences. This catalog does not have family names or allow you to search sequences (see below) - it provides direct access to individual protein sequences and can be searched by keyword.

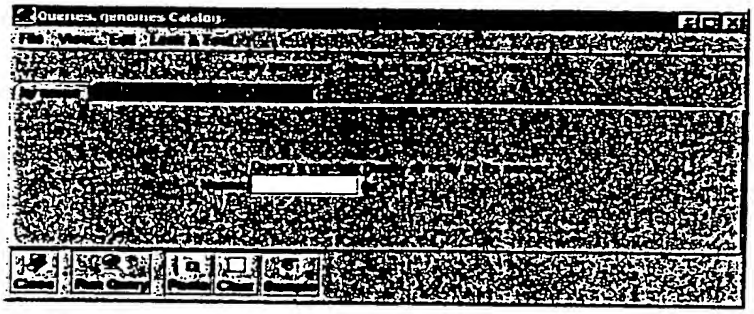


FIGURE 8

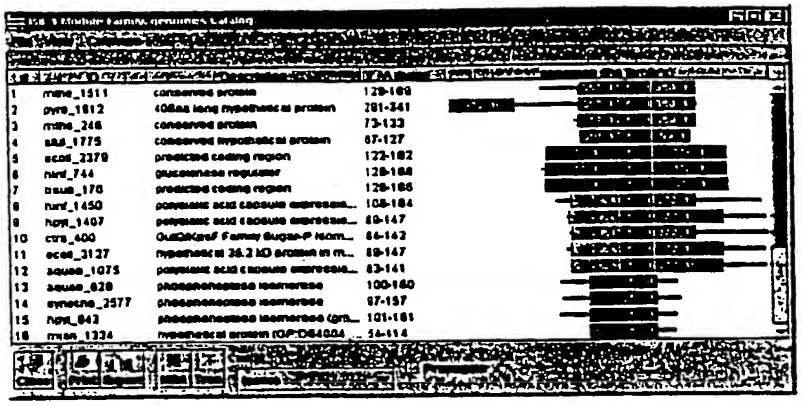
Query Window

The query window provides several different ways to search the chosen catalog: by catalog family identifier, by keyword, and by sequence. Not all of these methods are available for every catalog.

1. Search by Family identifier

Each family in a catalog has a unique identifier that defines a particular family. (These family names are generated automatically when each catalog is built from the protein sequence data.) This search window allows you to obtain a specific family number of interest. The result of this search is the family window showing a graphical view of the associated proteins and their modules.

FIGURE 9



140

Family window

The family window is the gateway to navigational power of MasterCatalog. The window shows all of the sequences in the currently selected catalog that are members of the that family. It shows the location of the modules in each of the sequences for that family (in red), and it also shows the location of all other modules in those sequence.

You can perform many different tasks from this point:

1. Display the multiple sequence alignment of the current family

Selecting the MSA button on the window's toolbar shows the multiple sequence alignment (the way in which the modules are related at the amino acid level)

2. Display the evolutionary tree of the current family

Selecting the Tree button on the window's shows the evolutionary tree. This indicates the pattern of divergence/similarity between individual modules, assuming that the distance between modules can be computed from the similarity in the protein sequences.

3. Display protein sequence information for any member of family

Double clicking on any sequence in the ID column shows the protein sequence window for that family (including catalog membership, description, and annotations).

4. Select sequences/modules

You can select some or all of the modules possessed by individual sequences. Modules are selected individually in the summary display with single mouse clicks. All of the modules possessed by a protein can be selected at once by selecting the module id (#) on the left hand column.

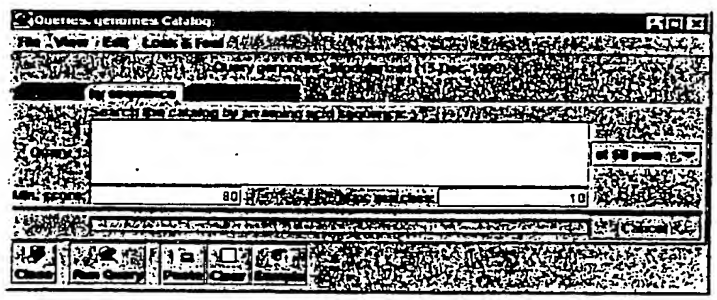
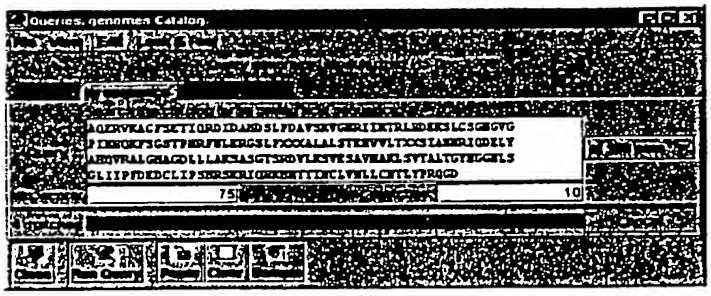


FIGURE 10

2. Search by Sequence

This window allows you to search a protein against a catalog for homologous (evolutionarily related) protein families. The result of this search is a sequence search summary window showing a graphical view of the relationship between the probe sequence and the associated families.
Note: This search can take as long as minutes for large probe sequences. You can abort the search at any time with the cancel button.



-125

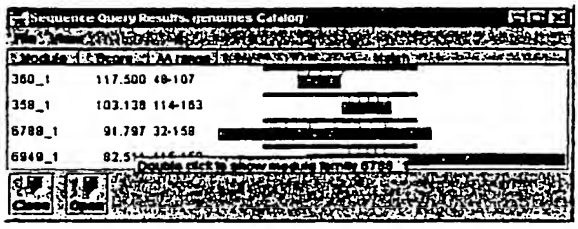
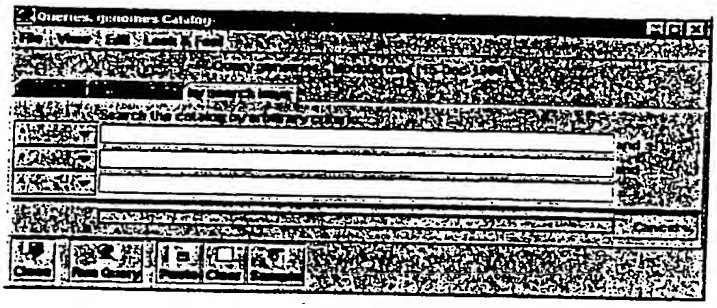


FIGURE 11

1. Sequence search results

This window shows the relationship between the probe sequence chosen for the query and each of the families that are related. The score corresponds to a 'log odds score', the probability that the relationship between the sequence is related according to the model of evolution vs the probability that the similarity is by chance.

Double clicking on any module shown in this summary will display the summary window for that family. The alignment of the probe sequence with the summary can then be displayed.



130

FIGURE 12

3. Search by Keyword

This window provides searches of the protein sequence database by keyword, according to annotations of proteins in the original sequence database. Keywords provided include selection by organism, by classification, by gene name and by gene product description. The result of this search is a sequence summary window showing a graphical view of individual protein sequences which fit the keyword search criteria.

Search results windows

These windows summarize the search results for search by sequence and search by keyword respectively.

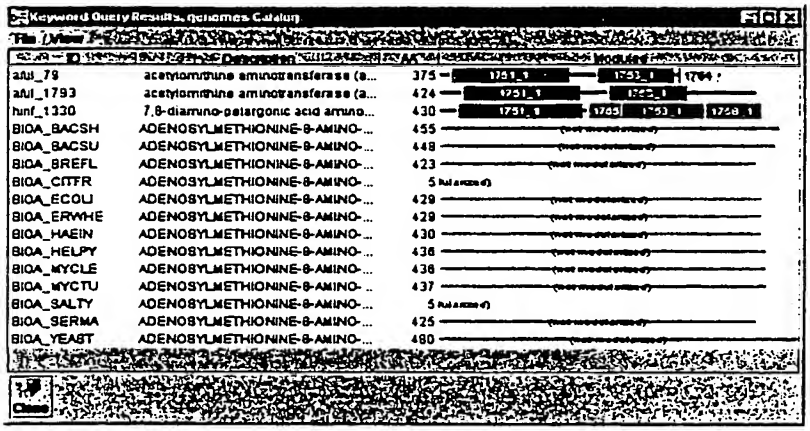
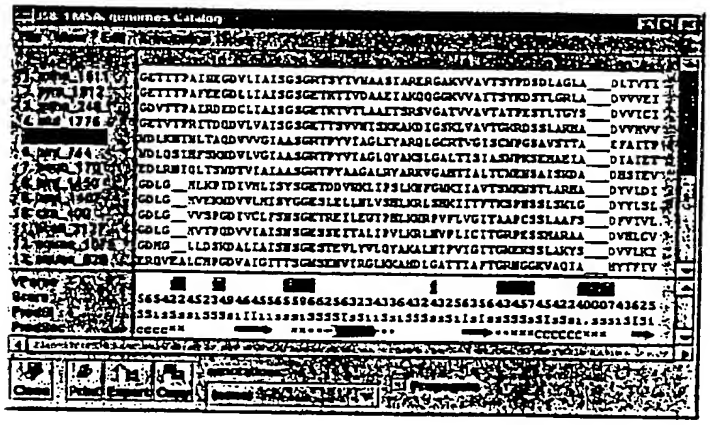


FIGURE 13

2. Keyword search results

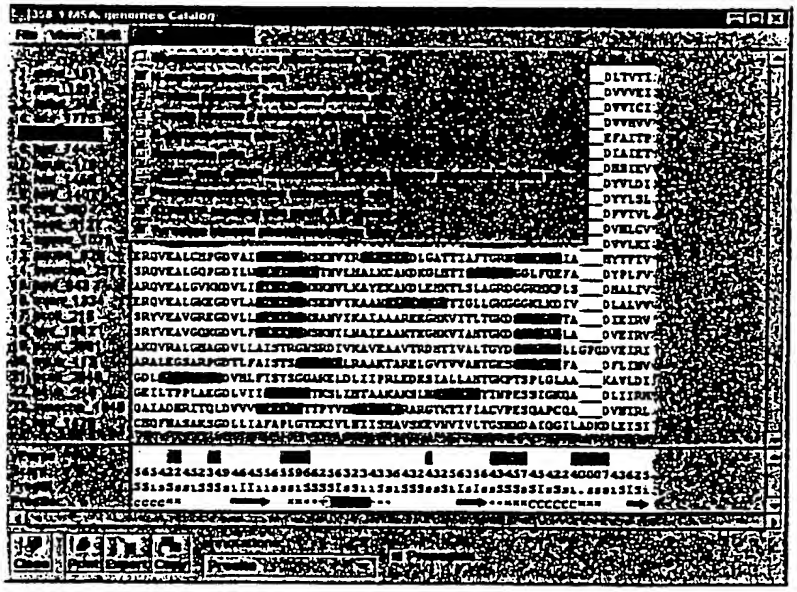
The keyword search results window shows the sequences. For each sequence all modules are shown for the currently selected catalog. Note: In some cases, sequences may match with a particular keyword but no modularization information is available because these sequences were not part of the set included in the currently selected catalog.

Double clicking on any module shown in the search results will display the summary window for the associated family.



150

FIGURE 14



150

17/23

FIGURE 15

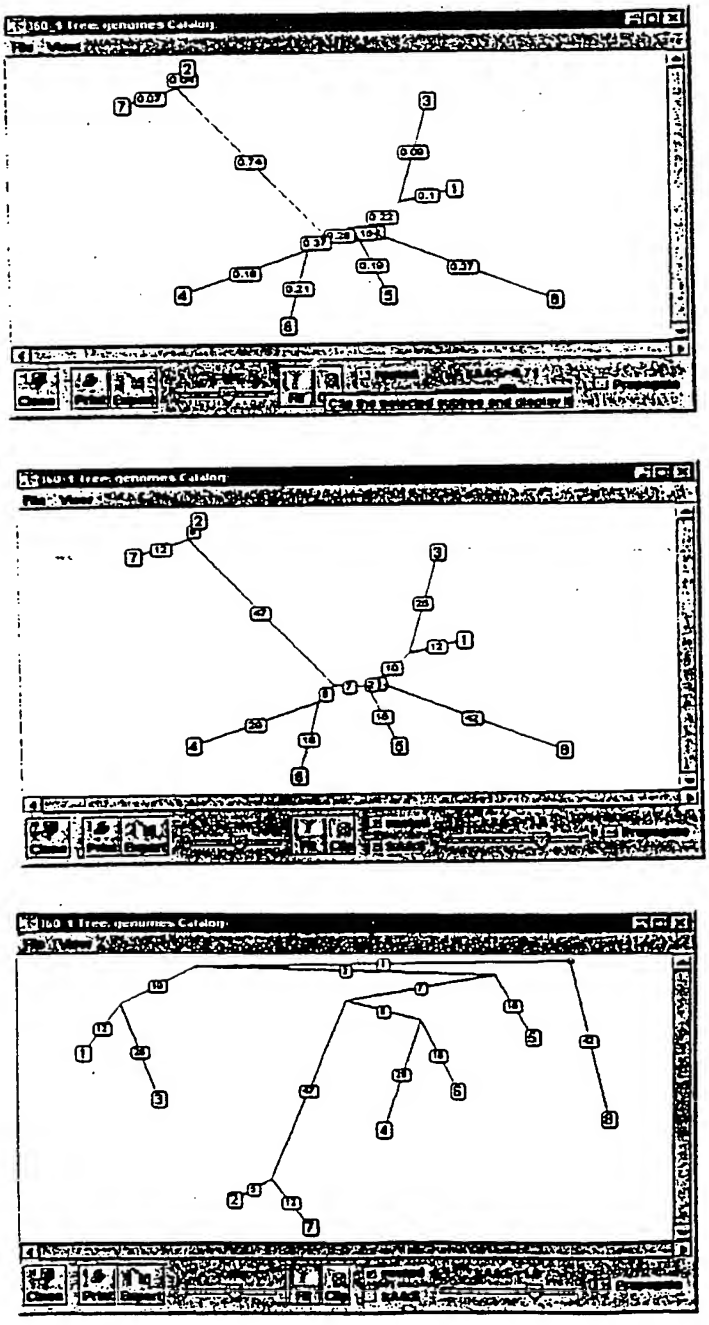


FIGURE 16

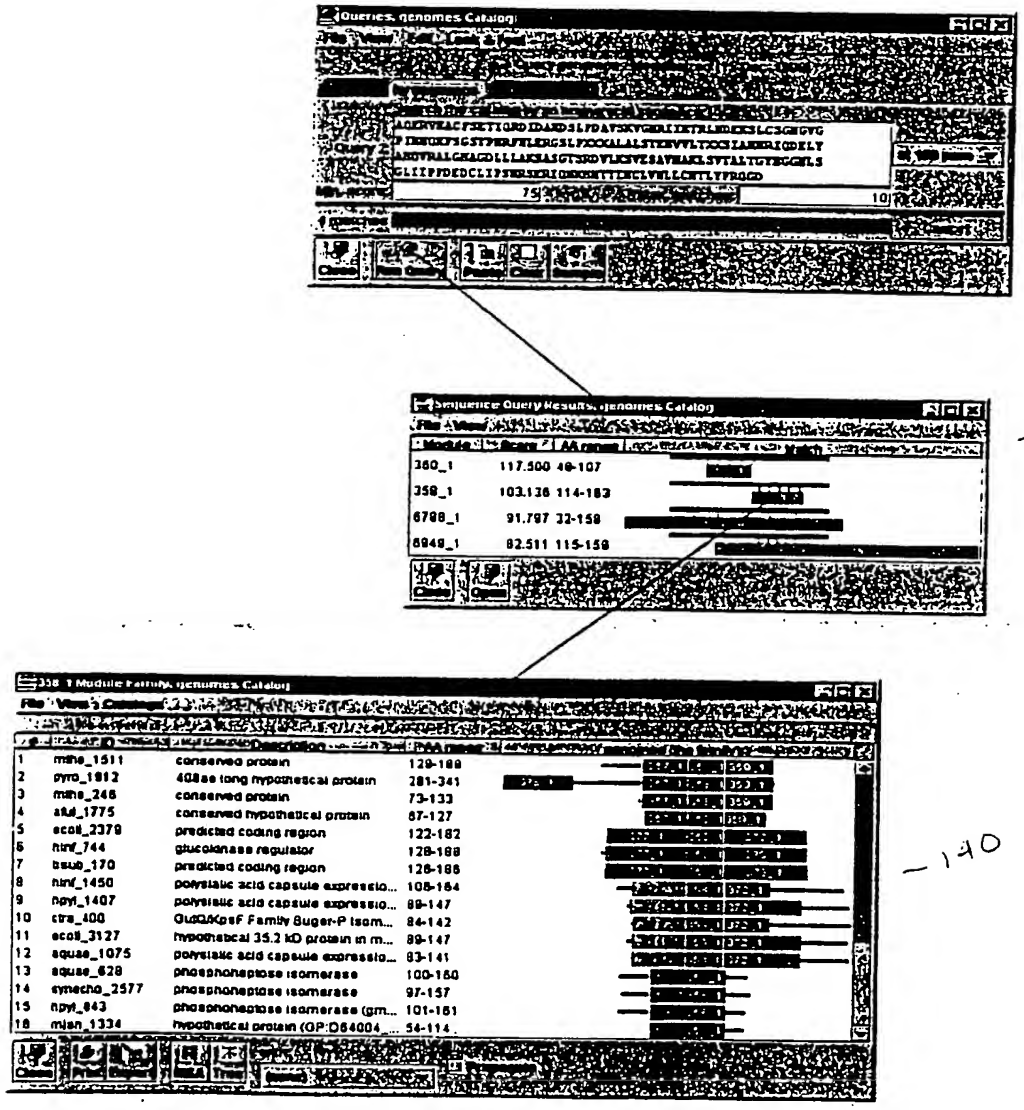

Queries, genomes Catalog
Sequence Query Results, genomes Catalog
360_1 117.500 46-107
359_1 103.136 114-163
6788_1 91.797 32-158
6949_1 82.511 115-158
1 mthe_1511 conserved protein 129-188
2 pyro_1812 408aa long hypothetical protein 281-341
3 mthe_246 conserved protein 73-133
4 aful_1775 conserved hypothetical protein 87-127
5 ecoli_2379 predicted coding region 122-182
6 hinf_744 gluconase regulator 128-188
7 tsub_170 predicted coding region 126-186
8 hinf_1450 polysialic acid capsule expressio... 105-164
9 npyl_1407 polysialic acid capsule expressio... 89-147
10 ctra_400 GlcNAcpF Family Sugar-P isom... 84-142
11 ecoli_3127 hypothetical 35.2 kD protein in m... 89-147
12 aquae_1075 polysialic acid capsule expressio... 83-141
13 aquae_628 phosphoneptose isomerase 100-160
14 synecho_2577 phosphoneptose isomerase 97-157
15 npyl_843 phosphoneptose isomerase (gm... 101-161
16 mjan_1334 hypothetical protein (GP:D64004_... 54-114

FIGURE 17

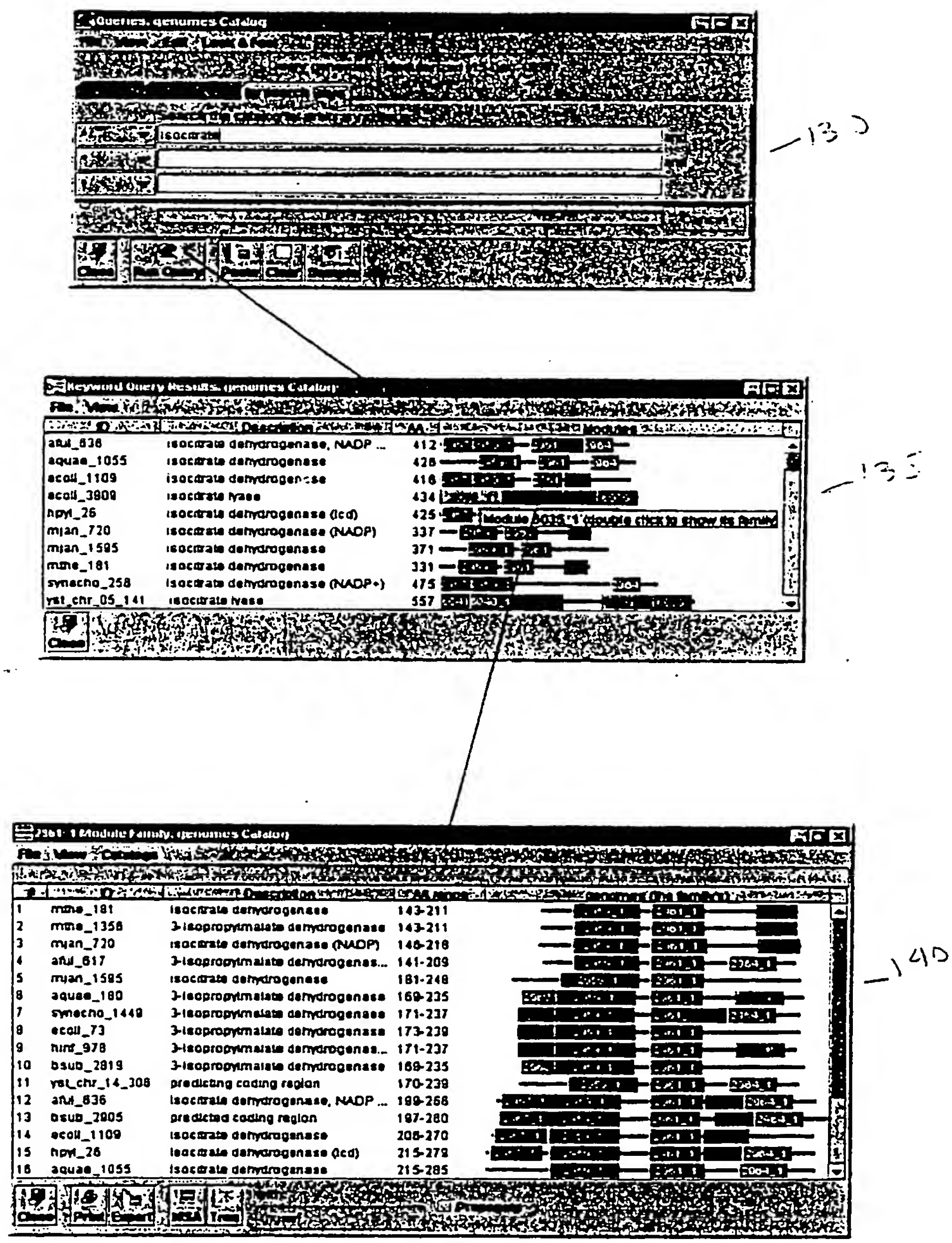
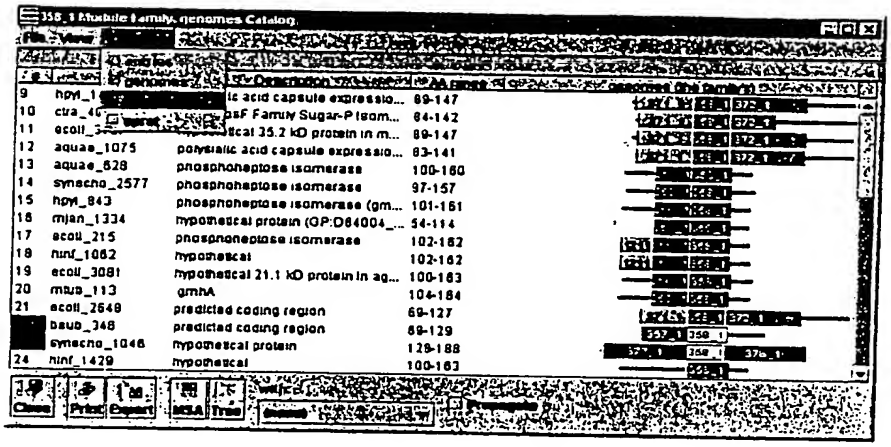

Queries. genomes Catalog
isocitrate
Close
Run Query
Paste
Clear
130
Keyword Query Results. genomes Catalog
File
View
Description
Modules
afu_636 isocitrate dehydrogenase, NADP ... 412
aquae_1055 isocitrate dehydrogenase 428
ecoli_1109 isocitrate dehydrogenase 416
ecoli_3909 isocitrate lyase 434
hpyl_26 isocitrate dehydrogenase (icd) 425
mjan_720 isocitrate dehydrogenase (NADP) 337
mjan_1595 isocitrate dehydrogenase 371
mthe_181 isocitrate dehydrogenase 331
synecho_258 isocitrate dehydrogenase (NADP+) 475
yst_chr_05_141 isocitrate lyase 557
Module 3035*1 (double click to show its family)
Close
135
2361*1 Module Family. genomes Catalog
File
View
Description
1 mthe_181 isocitrate dehydrogenase 143-211
2 mthe_1358 3-isopropylmalate dehydrogenase 143-211
3 mjan_720 isocitrate dehydrogenase (NADP) 148-216
4 afu_617 3-isopropylmalate dehydrogenas... 141-209
5 mjan_1595 isocitrate dehydrogenase 181-249
6 aquae_180 3-isopropylmalate dehydrogenase 169-235
7 synecho_1449 3-isopropylmalate dehydrogenase 171-237
8 ecoli_73 3-isopropylmalate dehydrogenase 173-239
9 hirf_978 3-isopropylmalate dehydrogenas... 171-237
10 bsub_2819 3-isopropylmalate dehydrogenase 169-235
11 yst_chr_14_308 predicting coding region 170-239
12 afu_636 isocitrate dehydrogenase, NADP ... 199-268
13 bsub_2805 predicted coding region 197-260
14 ecoli_1109 isocitrate dehydrogenase 208-270
15 hpyl_26 isocitrate dehydrogenase (icd) 215-279
16 aquae_1055 isocitrate dehydrogenase 215-285
Close
Print
Export
MSA
Tree
140

FIGURE 18A



140

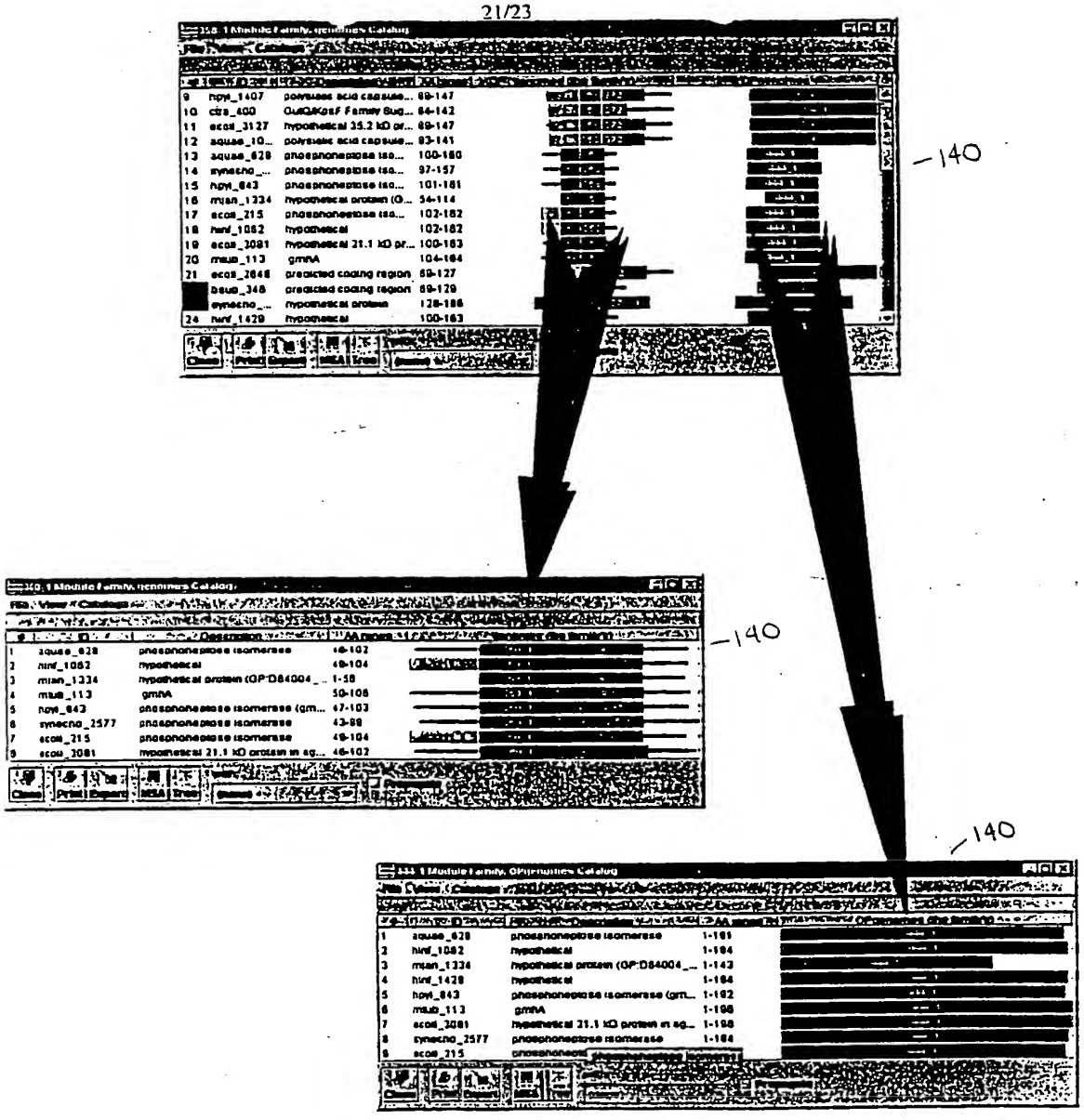


FIGURE 18B

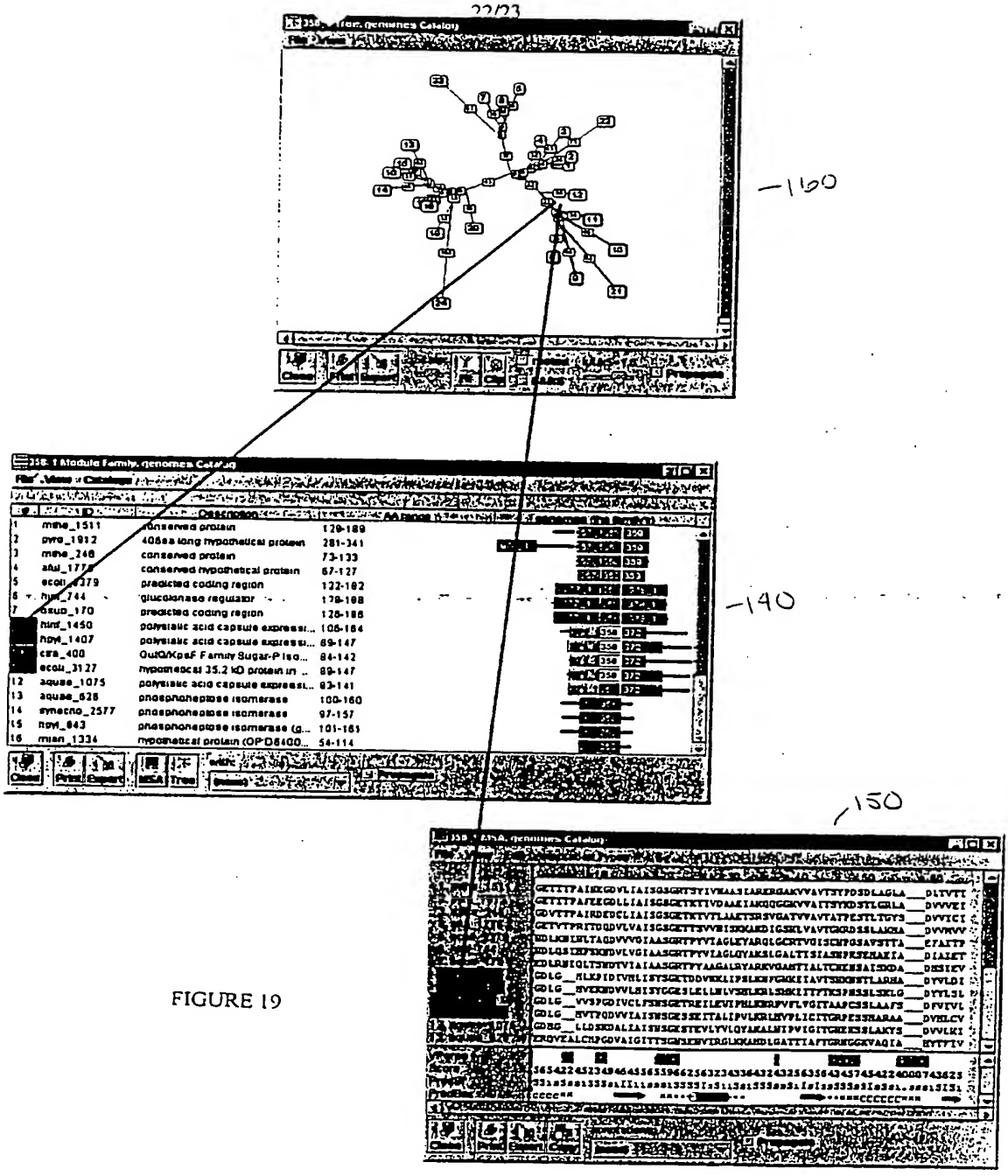


FIGURE 19

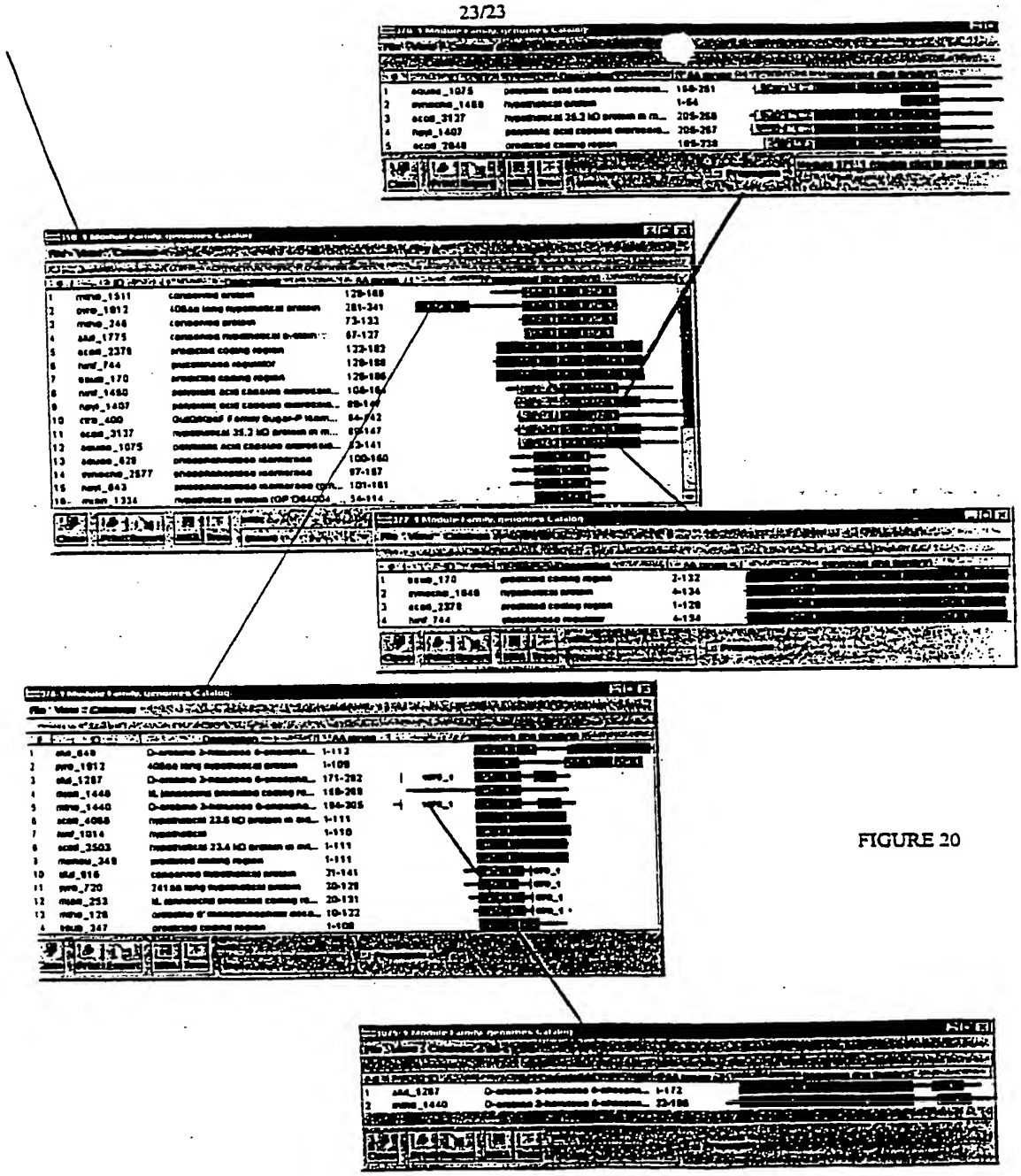


FIGURE 20

APPENDIX A

Saliwanchik, Lloyd & Saliwanchik, P.A.
2421 N.W. 41st Street, Suite A-1
Gainesville, FL 32606
(352) 375-8100

```
package mastercatalog.util;

import java.util.*;

/**
  The class managing and propagating selections of indices per id. Global
  propagation is possible through maps of indices to global ids.
  @author Lukas Knecht
*/

/*
  $Id: IndexSelection.java,v 1.5 1999/08/19 11:25:46 knecht Exp $
  $Log: IndexSelection.java,v $
  Revision 1.5  1999/08/19 11:25:46  knecht
  Added hasSelected() and getSubsetName().

  Revision 1.4  1999/07/30 08:43:13  knecht
  Added global propagation of selections.

  Revision 1.3  1999/07/19 08:30:09  knecht
  Added multiple toggle.

  Revision 1.2  1999/04/26 12:13:00  knecht
  Modifications for V2.0

  Revision 1.1  1999/02/18 21:42:16  knecht
  Initial revision

*/
public class IndexSelection {
  private static Hashtable selectionTable = new Hashtable();
  private static boolean globalPropagation;
  private BitSet selected;
  private boolean changed; // true if anything changed since the last propagate()
  private Vector listeners;
  private int[] globalIds;    maps index to ids for global propagation

  /**
    Create a new empty IndexSelection with a given id.
    @param id the id uniquely identifying this selection.
    */
  protected IndexSelection(int id) {
    selected = new BitSet(); changed = false; listeners = new Vector();
    selectionTable.put(new Integer(id), this);
  }

  /**
    Obtain the index selection belonging to a given id. This is the only
    method to create an IndexSelection.
    @param id the id uniquely identifying the selection. If no IndexSelection
    with this id exists, it is created.
    @return the IndexSelection belonging to the given id.
    */
  public static IndexSelection obtainIndexSelection(int id) {
    IndexSelection selection = getIndexSelection(id);
    if (selection == null) selection = new IndexSelection(id);
    return selection;
  }

  /**
    Get index selection belonging to a given id.
    @param id the id uniquely identifying the selection.
    @return the IndexSelection belonging to the given id or null if no such
    IndexSelection exists.
    */
  public static IndexSelection getIndexSelection(int id) {
    return (IndexSelection)selectionTable.get(new Integer(id));
  }

  /**
    Enable or display global propagation of selections.
    @param enabled true if global propagation should be enabled, false otherwise.
```

```
    */
  public static void setGlobalPropagation(boolean enabled) {
    globalPropagation = enabled;
  }

  /**
    Get the status of global propagation of selections.
    @return true if enabled, false otherwise.
    */
  public static boolean getGlobalPropagation() { return globalPropagation; }

  /**
    Set the mapping from indices to global ids for global propagation of
    the selection. If global propagation is enabled, copies existing selections.
    @param indexToId the mapping. Setting it to null disables global propagation
    from this IndexSelection.
    */
  public void setIndexToId(int[] indexToId) {
    this.indexToId = indexToId;
    if (globalPropagation && indexToId != null)
      for (Enumeration e = selectionTable.elements(); e.hasMoreElements();) {
        IndexSelection other = (IndexSelection)e.nextElement();
        if (other != this && other.indexToId != null) {
          for (int i = 0; i < indexToId.length; i++)
            for (int j = 0; j < other.indexToId.length; j++)
              if (indexToId[i] == other.indexToId[j])
                if (other.isSelected(j)) select(i);
                else deselect(i);
        }
      }
  }

  /**
    Get the mapping from indices to global ids for global propagation of
    the selection.
    @return the mapping.
    */
  public int[] getIndexToId() { return indexToId; }

  /**
    Get the current selection.
    @return the set of currently selected indices.
    */
  public BitSet getSelected() { return selected; }

  /**
    Add a listener listening to selection changes.
    @param l the listener to be added.
    */
  public synchronized void addSelectionListener(IndexSelectionListener l) {
    if (!listeners.contains(l)) listeners.addElement(l);
  }

  /**
    Remove a previously added listener.
    @param l the listener to be removed.
    */
  public synchronized void removeSelectionListener(IndexSelectionListener l) {
    listeners.removeElement(l);
  }

  /**
    Select the given index and note any change.
    @param index the index to be selected.
    */
  public synchronized void select(int index) {
    if (!selected.get(index)) { selected.set(index); changed = true; }
  }

  /**
    Select the given indices and note any change.
    @param indices the indices to be selected.
```

```
   */
  public synchronized void select(int[] indices) {
    for (int i = 0; i < indices.length; i++) select(indices[i]);
  }

  /**
    Deselect the given index and note any change.
    @param index the index to be deselected.
    */
  public synchronized void deselect(int index) {
    if (selected.get(index)) { selected.clear(index); changed = true; }
  }

  /**
    Deselect the given indices and note any change.
    @param indices the indices to be deselected.
    */
  public synchronized void deselect(int[] indices) {
    for (int i = 0; i < indices.length; i++) deselect(indices[i]);
  }

  /**
    Toggle the selection of the given index, optionally allowing multiple
    selections.
    @param index the index to be toggled.
    @param multiple allows for multiple selections: if false and index is being
    selected, deselects all other indices.
    */
  public synchronized void toggle(int index, boolean multiple) {
    if (selected.get(index)) selected.clear(index);
    else {
      if (!multiple) selected = new BitSet();
      selected.set(index);
    }
    changed = true;
  }

  /**
    Toggle the selection of the given indices, optionally allowing multiple
    selections.
    @param indices the indices to be toggled.
    @param multiple allows for multiple selections: if false and any index is
    being selected, deselects all other indices.
    */
  public synchronized void toggle(int[] indices, boolean multiple) {
    int i;
    BitSet newSelected = selected;
    if (!multiple) {
      for (i = 0; i < indices.length && selected.get(indices[i]); i++);
      if (i < indices.length) newSelected = new BitSet();
    }
    for (i = 0; i < indices.length; i++)
      if (selected.get(indices[i])) newSelected.clear(indices[i]);
      else newSelected.set(indices[i]);
    selected = newSelected; changed = true;
  }

  /**
    Get the selection for a given index.
    @param index the index to be queried.
    @return true if index is selected, false otherwise.
    */
  public synchronized boolean isSelected(int index) {
    return selected.get(index);
  }

  /**
    Check whether there is any selection at all.
    @return true if any index is selected, false otherwise.
    */
  public synchronized boolean hasSelected() {
    int i;
```

```
      for (i = 0; i < selected.size() && !selected.get(i); i++);
      return i < selected.size();
    }

    /**
      Propagate the current selection state to the SelectionListeners if it
      changed since the last propagation.
      @param source the listener which should not be notified.
      */
    public synchronized void propagate(IndexSelectionListener source) {
      if (changed) {
        changed = false;
        for (int i = 0; i < listeners.size(); i++) {
          IndexSelectionListener l =
            (IndexSelectionListener)listeners.elementAt(i);
          if (l != source) l.selectionChanged(this);
        }
        propagateGlobally();
      }
    }

    /**
      If global propagation is enabled, propagate the current selection state
      globally to all other selections.
      */
    public synchronized void propagateGlobally() {
      if (globalPropagation && indexToId != null)
        for (Enumeration e = selectionTable.elements(); e.hasMoreElements();) {
          IndexSelection other = (IndexSelection)e.nextElement();
          if (other != this && other.indexToId != null) {
            for (int i = 0; i < indexToId.length; i++)
              for (int j = 0; j < other.indexToId.length; j++)
                if (indexToId[i] == other.indexToId[j])
                  if (isSelected(i)) other.select(j);
                  else other.deselect(j);
            other.propagate(null);
          }
        }
    }

    /**
      Get a simple standard name for a BitSet.
      @param subset the set. May be null.
      @return a (nonunique) name corresponding to the subset.
      */
    public static String getSubsetName(BitSet subset) {
      if (subset == null) return "null";
      int mini = 1, maxi = 0;
      for (int i = 0; i < subset.size(); i++)
        if (subset.get(i)) {
          if (mini > maxi) mini = i;
          maxi = i;
        }
      return "[" + mini + "," + maxi + "]";
    }
  }
```

```
package mastercatalog.gui;

import mastercatalog.gui.event.*;
import mastercatalog.ds.*;
import mastercatalog.util.*;

import java.io.*;
import java.sql.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;

/**
  The standard frame showing the data about a family as a table.
  @author Lukas Knecht
*/

/*
  $Id: FamilyFrame.java,v 1.10 1999/08/19 11:25:45 knecht Exp $
  $Log: FamilyFrame.java,v $
  Revision 1.10  1999/08/19 11:25:45  knecht
  Various GUI changes.

  Revision 1.9  1999/08/10 13:55:11  knecht
  Change for new busy indicator.

  Revision 1.8  1999/07/30 08:43:13  knecht
  Various changes and modifications.

  Revision 1.7  1999/07/19 09:44:10  knecht
  No tree and no MSA give error message.

  Revision 1.6  1999/07/19 08:16:20  knecht
  Various GUI changes.

  Revision 1.5  1999/04/26 10:13:19  knecht
  Modifications for V2.0

  Revision 1.4  1999/02/19 10:08:26  knecht
  Added menus.

  Revision 1.3  1999/02/18 21:45:02  knecht
  Various big changes and enhancements for V1.0

  Revision 1.2  1998/12/14 22:19:28  sgc
  Various bug fixes and enhancements, including the export function and MSA ordering

  Revision 1.1  1998/10/02 14:15:46  knecht
  Initial revision

*/
public class FamilyFrame extends MCFrame
implements Exportable, Printable, IndexSelectionListener, CellClickListener {
  private static final int [illegible] = 0;
  private static final int [illegible] = 1;
  private static final String[] [illegible] = { [illegible] };
  private static FrameTable openFrames = new FrameTable();
  private Family family;
  private BitSet subset;
  private String name;
  private FamilyTableModel model;
  private MCTable table;
  private IndexSelection selection;
  private BitSet oldSelection;
  private int[] memberToSequence;
  private FamilySequence[] extraSequences;
  private JComboBox searches; // the list of available searches

  /**
    Create a frame showing a family belonging to a certain catalog.
    @param server the ServerConnection to get the available catalogs from.
```

```
    @param catalog the catalog the family belongs to.
    @param family the family to be shown.
    @param subset a BitSet identifying the subset. May be null.
    */
  public FamilyFrame(ServerConnection server, Family family, BitSet subset)
      throws Exception
  {
    super(server,
    this.family = family; this.subset = subset;
    String subsetName = IndexSelection.getSubsetName(subset);
    name = family.getId() +  + subsetName; openFrames.put(name, this);
    Catalog catalog = server.getCatalogById(family.catId);
    setTitle(family.getFullName() + subsetName +
            catalog.name +  );
    memberToSequence = family.getMemberToSequence();

    // Listen to changes in the selection
    selection = IndexSelection.obtainIndexSelection(family.getId());
    selection.setIndexToId(family.getMemberToSequenceId());
    selection.addSelectionListener(this);

    table = new MCTable();
    JTextField descr = new JTextField();
    descr.setEditable(false);
    if (Capabilities.areAllToolTipsSupported())
      descr.setToolTipText(
    add(descr,          );
    add(new JScrollPane(table),          );
    model = new FamilyTableModel(catalog, family);
    table.setModel(model);
    table.setContext(server, catalog);
    table.setSelection(selection);
    oldSelected = (BitSet)selection.getSelected().clone();
    table.addCellClickListener(this);
    addStandardMenuItems(catalog);

    toolBar.addSeparator();
    final SlowAction msaAction = new SlowAction() {
      public void act() { showMSA(); }
    };
    MCAbstractAction msa =
      new MCAbstractAction(

      public void actionPerformed(ActionEvent e) { msaAction.run(); }
    );
    addToToolBar(msa); addToViewMenu(msa,  );

    final SlowAction treeAction = new SlowAction() {
      public void act() { showTree(); }
    };
    MCAbstractAction tree =
      new MCAbstractAction(

      public void actionPerformed(ActionEvent e) { treeAction.run(); }
    );
    addToToolBar(tree); addToViewMenu(tree,  );
    MCAbstractAction clip =
      new MCAbstractAction(

      {
        public void actionPerformed(ActionEvent e) { clipSelected(); }
      };
    addToToolBar(clip); addToViewMenu(clip,  );

    JMenu catMenu = new JMenu(
    addToMenuBar(catMenu);
    Catalog[] allCats = server.getCatalogs();
    for (int i = 0; i < allCats.length; i++) {
      JCheckBoxMenuItem item = new JCheckBoxMenuItem(allCats[i].name);
      item.setEnabled(!allCats[i].isEntryCatalog() &&
                      allCats[i].getId() != family.catId);
      item.setState(allCats[i].getId() == family.catId);
```

```
      item.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          toggleCatalog(((JMenuItem)e.getSource()).getText());
        }
      });
      addToCurrentMenu(item);
    }
    catMenu.setEnabled(server.isXNavigationAllowed());

    toolBar.addSeparator();
    JPanel seqPanel = new JPanel(new GridLayout(2, 1));
    seqPanel.setMaximumSize(new Dimension(150, 100));
    seqPanel.add(new JLabel(
    searches = new JComboBox();
    searches.setToolTipText(
    seqPanel.add(searches); toolBar.add(seqPanel);
    toolBar.addSeparator(); addPropagateBoxes(selection);
    table.setRowSelectionAllowed(
    descr.setText(family.descr);
  }

  /**
    Show the Tree belonging to this family.
   */
  private void showTree() {
    String
    int    = searches.getSelectedIndex();
    if (i > 0) search = (String)searches.getItemAt(i);
    TreeFrame
    if (frame !=
  }

  /**
    Show the MSA belonging to this family.
   */
  private void showMSA() {
    String
    int    = searches.getSelectedIndex();
    if (i > 0) search = (String)searches.getItemAt(i);
    MSAFrame
    if (frame !=
  }

  /**
    Open a new frame with the selected modules.
    Does nothing if there are no selected modules.
   */
  private void clipSelected() {
    if (selection.hasSelected()) {
      FamilyFrame
        FamilyFrame.openFrame(server, family.getId(), selection.getSelected());
      if (clipFrame !=
    }
  }

  /**
    Toggle a column displaying a given catalog.
    @param name the name of the catalog.
   */
  private void toggleCatalog(final String name) {
    try {
      Catalog
      if (cat !=
        int
        if (i >= 0) {
          model.removeCatalog(cat); table.removeColumn(i);
        }
        else {
          FamilySequence[]
            new FamilySequence[family.sequence.length];
          for (i = 0; i < sequence.length; i++)
            sequence[i] =
```

```
                server.getFamilySequence(cat.getId(), family.sequence[i].id);
              model.addCatalog(cat, sequence);
              table.addColumn(FamilyTableModel.SEQUENCE_WIDTH);
            }
          }
        }
        catch (Exception e) { UserMessages.show(e); }
      }

      /**
       Get the matching search ids for this family.
       */
      private void getSearches() {
        if (searches.getItemCount() != 0) searches.removeAllItems();
        searches.addItem("          ");
        String[] names = server.getSearchNames(family);
        for (int i = 0; i < names.length; i++) searches.addItem(names[i]);
        searches.setEnabled(names.length > 0);
        extraSequence = server.getSearchSequences(family);
        int[] extraIndent = server.getSearchSequenceIndents(family);
        model.setExtraSequences(extraSequence, extraIndent);
      }

      public void dispose() {
        openFrames.remove(name); super.dispose();
      }

      /**
       Apply the user preferred properties to this frame.
       @param withSize apply also the size property.
       */
      public void applyProperties(boolean withSize) {
        table.applyProperties(withSize);
        if (withSize) pack();
      }

      /**
       Get all child frames.
       */
      public MCFrame[] getChildren() {
        MCFrame[] trees = TreeFrame.getAllWithPrefix(family.getId() + "/");
        MCFrame[] msas = MSAFrame.getAllWithPrefix(family.getId() + "/");
        MCFrame[] res = new MCFrame[trees.length + msas.length];
        System.arraycopy(trees, 0, res, 0, trees.length);
        System.arraycopy(msas, 0, res, trees.length, msas.length);
        return res;
      }

      /**
       Write the parameters to create this frame to a serializer.
       @param out the serializer to write to.
       */
      public void writeParameters(ASCIISerializer out) throws Exception {
        Catalog catalog = server.getCatalogById(family.catId);
        out.writeQuoted(catalog.name); out.write(" ");
        out.writeQuoted(family.getFullName());
        if (subset != null) { out.write(" "); out.write(subset); }
      }

      /**
       Create a frame to show a given family if it does not already exist,
       make it visible and put it on top of the window stack.
       @param server the ServerConnection to use.
       @param famId the id of the family to be shown.
       @param subset a BitSet defining a subset of modules to be displayed.
       May be null.
       */
      public static FamilyFrame openFrame(ServerConnection server, int famId,
                                          BitSet subset) {
        String subsetName = IndexSelection.getSubsetName(subset);
        FamilyFrame frame = (FamilyFrame)openFrames.get(famId + "." + subsetName);
        if (frame == null) {
```

```
      try {
        Family family = server.getFamily(famId);
        if (subset != null)
          family = family.cloneSelected(subset);
        frame = new FamilyFrame(server, family, subset);
      }
      catch (Exception e) { UserMessages.show(e); }
    }
    if (frame != null) {
      frame.getSearches(); frame.applyProperties(true); frame.setVisible(true);
      frame.toFront();
    }
    return frame;
  }

  /**
    Create a frame from a command read from a deserializer.
    The string has the format "catalog '.' family".
    @param server the ServerConnection to use.
    @param in the deserializer to read the command from.
    */
  public static FamilyFrame openFrame(ServerConnection server,
                                      ASCIIDeserializer in)
      throws Exception
  {
    String name = in.readString();
    Catalog catalog = server.getCatalogByName(name);
    if (catalog == null) {
      UserMessages.show("Unknown catalog: " + name);
      return null;
    }
    in.checkChar('.');
    name = in.readString();
    int familyId = server.getFamilyId(catalog, name);
    if (familyId < 0) {
      UserMessages.show("Unknown family: " + name);
      return null;
    }
    BitSet subset = null;
    if (in.ttype == '[') { in.nextToken(); subset = in.readBitSet(); }
    return openFrame(server, familyId, subset);
  }

  /**
    Get the history text corresponding to the command read from a deserializer.
    @param in the deserializer to read the command from.
    */
  public static String getHistoryText(ASCIIDeserializer in)
      throws IOException, SyntaxException
  {
    StringBuffer text = new StringBuffer(100);
    String catalog = in.readString(); in.checkChar('.');
    String family = in.readString();
    text.append(family);
    if (in.ttype == '[') {
      in.nextToken(); text.append('['); text.append(in.readBitSet().toString());
    }
    text.append(" in "); text.append(catalog);
    return text.toString();
  }

  /**
    Dispose all open frames.
    */
  public static void disposeAll() { openFrames.disposeAll(); }

  /**
    Get all open frames.
    */
  public static MCFrame[] getAll() { return openFrames.getAll(); }

  /**
```

```
  Apply the properties to all open frames.
  @param withSize true if the frame size should be set.
  */
public static void applyPropertiesToAll(boolean withSize) {
  openFrames.applyPropertiesToAll(withSize);
}

/* The Printable implementation: */

public void print(PrintJob job, Rectangle area) {
  Graphics g = job.getGraphics();
  Font ssFont = Fontbase.getSansSerif( : 10 , 10);
  g.setFont(ssFont);
  int fontSize =
    server.getProperties().getIntProperty(ClientProperties.MODULE_FONTSIZE);
  Font moduleFont = Fontbase.getSansSerif( ... , fontSize);
  FontMetrics fm = g.getFontMetrics();
  // compute labels and ranges and the width of the first three columns
  String[] label = new String[family.sequence.length];
  String[] range = new String[family.sequence.length];
  int dxLabel = 0;
  int dxId = 0;
  int dxRange = 0;
  for (int i = 0; i < family.sequence.length; i++) {
    Module[] modules =
      family.sequence[i].getFamilyModulesByIndex(family.getId());
    StringBuffer buf = new StringBuffer();
    for (int j = 0; j < modules.length; j++) {
      if (j > 0) buf.append( ');
      buf.append(modules[j].index);
    }
    label[i] = buf.toString();
    buf = new StringBuffer( );
    for (int j = 0; j < modules.length; j++) {
      if (j > 0) buf.append( );
      buf.append(modules[j].seqStart + 1);
      buf.append( - );
      buf.append(modules[j].seqStart - modules[j].seqLength);
    }
    range[i] = buf.toString();
    int w = fm.stringWidth(label[i]);
    if (w > dxLabel) dxLabel = w;
    w = fm.stringWidth(family.sequence[i].name);
    if (w > dxId) dxId = w;
    w = fm.stringWidth(range[i]);
    if (w > dxRange) dxRange = w;
  }
  // compute column positions
  int space = fm.charWidth( );
  int xId = area.x + dxLabel + space;
  int xRange = xId + dxId + space;
  int xSeq = xRange + dxRange + space;
  int wSeq = area.width - xSeq;
  // compute the number of lines
  int chHeight = fm.getHeight() + fm.getMaxDescent();
  int lines = (area.height / chHeight - 1) / 2 - 2; // count title
  for (int i0 = 0; i0 < family.sequence.length; i0 += lines) {
    int i1 = i0 + lines;
    if (i1 > family.sequence.length) i1 = family.sequence.length;

    // create a new page if none exists yet and print the title
    if (g == null) g = job.getGraphics();
    g.setFont(Fontbase.getSansSerif( , 14));
    g.drawString(getTitle() + ' ' + (i0 + 1) + ' ' + i1,
                 area.x, area.y + chHeight);

    // print the sequences
    int y = area.y + 2 * chHeight;
    for (int i = i0; i < i1; i++) {
      FamilySequence sequence = family.sequence[i];
      y += chHeight;
      // print label, id and range
```

```
        g.setFont(ssFont); g.setColor(Color.black);
        DrawingUtilities.
          drawRightAdjustedString(g, label[i], area.x + dxLabel, y);
        g.drawString(sequence.name, xId, y);
        g.drawString(range[i], xRange, y);
        // print gray bar representing the sequence
        g.setColor(Color.gray);
        g.fillRect(xSeq + family.indent[i] * dxSeq / family.maxLength,
                   y - 4 * chHeight / 10,
                   sequence.length * dxSeq / family.maxLength,
                   2 * chHeight / 10);
        g.setFont(moduleFont);
        for (int j = 0; j < sequence.module.length; j++)
          DrawingUtilities.drawModule
            (g, xSeq + (family.indent[i] + sequence.module[j].seqStart)
             * dxSeq / family.maxLength, y - 9 * chHeight / 10,
             sequence.module[j].seqLength * dxSeq / family.maxLength,
             8 * chHeight / 10, sequence.module[j].famName,
             sequence.module[j].famId == family.getId(), true);
      }

      // print the descriptions
      y += chHeight / 2;
      for (int i = i0; i < i1; i++) {
        FamilySequence sequence = family.sequence[i];
        y += chHeight;
        // print label, id and description
        g.setFont(ssFont); g.setColor(Color.black);
        DrawingUtilities.
          drawRightAdjustedString(g, label[i], area.x - dxLabel, y);
        g.drawString(sequence.name, xId, y);
        g.drawString(sequence.descr, xRange, y);
      }

      // print this page and force allocation of the next page
      g.dispose(); g = null;
    }
  }

  /* The IndexSelectionListener implementation: */

  /* A complete update of the entire column is too slow. We therefore only
     update the changed rows, which means that all selection changes in the
     table must happen through here to keep oldSelected consistent.
     */
  public void selectionChanged(IndexSelection selection) {
    BitSet selected = selection.getSelected();
    for (int i = 1; i < memberToSequence.length + extraSequence.length; i++)
      if (selected.get(i) != oldSelected.get(i)) {
        int row;
        if (i >= memberToSequence.length)
          row = i - memberToSequence.length + family.sequence.length;
        else
          row = memberToSequence[i];
        table.tableChanged(new TableModelEvent(model, row, row));
      }
    oldSelected = (BitSet)selected.clone();
  }

  /* The CellClickListener implementation: */

  public void cellClicked(CellClickEvent event) {
    if (event.getClickCount() == 2 && event.getColumn() == 1) {
      int row = event.getRow();
      if (row < family.sequence.length) {
        SequenceFrame frame =
          SequenceFrame.openFrame(server, family.sequence[row].id);
        if (frame != null) frame.appendToHistory();
      }
    }
  }
}
```

```
    /* The Exportable interface implementation: */

    public String[] getExportFormats() { return exportFormats; }

    public void export(Writer out, int format)
        throws IOException
    {
      switch (format) {
      case [illegible]:
        out.write(                    + family.getFullName() +     );
        for (int i = 0; i < family.sequence.length; i++) {
          FamilySequence seq = family.sequence[i];
          out.write(          + seq.name +      );
          for (int j = 0; j < seq.module.length; j++) {
            Module mod = seq.module[j];
            out.write(          + mod.famName +
                              + (mod.seqStart + 1) +
                              + (mod.seqStart + mod.seqLength) +
                            );
          }
          out.write(      );
        }
        break;
      case [illegible]:
        out.write(family.toString());
        break;
      }
    }
}
```

```
package mastercatalog.gui;

import mastercatalog.ns.*;
import mastercatalog.gui.event.*;

import java.util.*;
import javax.swing.table.*;

/**
  The table model responsible for repesentation of family information with
  sequences and their modules.
  @author: Lukas Knecht
  @version: 08-02-1998
*/

/*
  SId: FamilyTableModel.java,v 1.6 1999/08/19 11:25:45 knecht Exp S
  SLog: FamilyTableModel.java,v S
  Revision 1.6  1999/08/19 11:25:45  knecht
  Various GUI changes.

  Revision 1.5  1999/07/30 08:43:13  knecht
  Various GUI changes and modifications.

  Revision 1.4  1999/07/19 08:16:36  knecht
  Various GUI changes.

  Revision 1.3  1999/04/26 10:13:19  knecht
  Modifications for V2.0

  Revision 1.2  1999/02/18 21:45:02  knecht
  Various big changes and enhancements for V1.0

  Revision 1.2  1998/12/14 22:19:28  sgc
  Various bug fixes and enhancements, including the export function and MSA ordering

  Revision 1.1  1998/10/02 14:15:46  knecht
  Initial revision

*/
public class FamilyTableModel extends MCAbstractTableModel
implements SequenceDisplayable
{
  /**
   The preferred width of the sequence columns.
   */
  public final static int [illegible] = 300;

  private Catalog [illegible]; // the main catalog to be displayed
  private Family [illegible]; // the main family to be displayed
  private String[] [illegible] = {
                                  [illegible]};
  private int[] [illegible] = {30, 100, 200, 150, 80, SEQUENCE_WIDTH};
  private Vector [illegible] = new Vector();
  private Vector [illegible] = new Vector();
  private Object[][] [illegible];
  private int[] [illegible];

  /**
    Construct a new family table model.
    @param cat the catalog the family belongs to.
    @param family the family to be displayed.
    */
  public FamilyTableModel(Catalog catalog, Family family) {
    setContext(catalog, family);
  }

  /**
    Set the family to be displayed.
    @param cat the catalog the family belongs to.
    @param family the family to be displayed.
    */
```

```
    public void setContext(Catalog catalog, Family family) {
      this.family = family; this.catalog = catalog;
      // convert the family into the data to be displayed
      data = new Object[family.sequence.length][];
      labels[5] = catalog.name + " [illegible]";
      for (int i = 0; i < data.length; i++)
        data[i] = getRow(family.getId(), family.sequence[i]);
      otherCatalogs.removeAllElements(); otherSequences.removeAllElements();
    }

    private Object[] getRow(int famId, FamilySequence sequence) {
      Module[] modules = sequence.getFamilyModulesByIndex(famId);
      int[] indices = new int[modules.length];
      for (int j = 0; j < modules.length; j++) indices[j] = modules[j].index;
      Object[] row = new Object[6];
      row[0] = new SelectableString(indices);
      row[1] = sequence.name;
      row[2] = sequence.descr;
      row[3] = sequence.getSpecies();
      StringBuffer buf = new StringBuffer();
      for (int j = 0; j < modules.length; j++) {
        if (j > 0) buf.append(     );
        buf.append(modules[j].seqStart + 1);
        buf.append(   );
        buf.append(modules[j].seqStart + modules[j].seqLength);
      }
      row[4] = buf.toString();
      row[5] = sequence;
      return row;
    }

    /**
     Set extra sequences to be displayed (sequences which do not really belong
     to the family, but which make some sense when added.<BR>
     This adds sequences only to the main catalog column.
     @param sequences the extra sequences to be added.
     @param indent the indent of the extra sequences to be added in the
     context of the family.
     */
    public void setExtraSequences(FamilySequence[] sequences, int[] indent) {
      if (extraData != null)
        fireTableRowsDeleted(data.length, data.length + extraData.length - 1);
      extraData = null;
      if (sequences != null) {
        extraData = new Object[sequences.length][]; extraIndent = indent;
        for (int i = 0; i < sequences.length; i++)
          extraData[i] = getRow(family.getId(), sequences[i]);
        fireTableRowsInserted(data.length, data.length + extraData.length - 1);
      }
    }

    /**
     Add a new column displaying another catalog. Adding a catalog which
     is already displayed is a no-op.
     @param cat the catalog to add.
     @param sequence the same sequences as displayed by this model, but with
     modules of cat.
     */
    public void addCatalog(Catalog cat, FamilySequence[] sequence) {
      if (indexOf(cat) < 0) {
        otherCatalogs.addElement(cat.name); otherSequences.addElement(sequence);
      }
    }

    /**
     Get the column index of a catalog or -1 if the catalog is not shown.
     @param cat the catalog to check for.
     @return the model column index of the catalog or -1.
     */
    public int indexOf(Catalog cat) {
      int i = otherCatalogs.indexOf(cat.name);
      if (i >= 0) i += data[0].length;
```

```
    return i;
  }

  /**
    Remove a column displaying another catalog. Removing a catalog which is
    not displayed is a no-op.
    @param cat the catalog to remove.
    */
  public void removeCatalog(Catalog cat) {
    int i = otherCatalogs.indexOf(cat.name);
    if (i >= 0) {
      otherCatalogs.removeElementAt(i); otherSequences.removeElementAt(i);
    }
  }

  /* The AbstractTableModel implementation. */

  public String getColumnName(int col) {
    if (col < labels.length) return labels[col];
    return (String)otherCatalogs.elementAt(col - labels.length);
  }

  /* The MCAbstractTableModel implementation. */

  public String getToolTipTextAt(int row, int col) {
    FamilySequence seq =
      row < data.length ? family.sequence[row] :
      (FamilySequence)extraData[row - data.length][5];
    if (col == 1) return seq.getKeys(SeqKeyGroup.IDENTIFIER_GROUP);
    if (col == 2) return seq.descr;
    if (col == 3) return seq.getSpecies();
    return null;
  }

  public int[] getColumnWidths() { return widths; }

  public int getColumnCount() {
    return super.getColumnCount() + otherCatalogs.size();
  }

  public int getRowCount() {
    if (extraData == null) return super.getRowCount();
    else return super.getRowCount() + extraData.length;
  }

  public Object getValueAt(int row, int col) {
    if (row < data.length) {
      int i = col - data[row].length;
      if (i < 0) return super.getValueAt(row, col);
      return ((FamilySequence[])otherSequences.elementAt(i))[row];
    }
    else {
      row -= data.length;
      if (col < extraData[row].length) return extraData[row][col];
      else return null;
    }
  }

  public Class getColumnClass(int col) {
    Class cls = null;
    Object value = getValueAt(0, col);
    if (value != null) cls = value.getClass();
    else {
      // Object.class does not work in all VMs
      try { cls = Class.forName("java.lang.Object"); }
      catch (ClassNotFoundException e) {} // ignore
    }
    return cls;
  }

  /* The SequenceDisplayable implementation. */
```

```
  public int getFamilyId() { return family.getId(); }

  public int getIndent(int row) {
    if (row < data.length) return family.indent[row];
    else return extraIndent[row - data.length];
  }

  public int getMaxLength() { return family.maxLength; }

  /**
    Get the family being displayed.
    @return the family.
    */
  public Family getFamily() { return family; }
}
```

```
package masterr[illegible].util;

import java.util.*;

/**
  The interface for a class which listens to index selection changes.
  @author Lukas Knecht
  @version 01-17-1999
*/

/*
  $Id: IndexSelectionListener.java,v 1.1 1999/02/18 21:42:29 knecht Rel $
  $Log: IndexSelectionListener.java,v $
  Revision 1.1  1999/02/18 21:42:29  knecht
  Initial revision

  */
public interface [illegible] {

  /**
    Notify a listener that an index selection has changed.
    @param [illegible] the IndexSelection which has changed.
    */
  public void selectionChanged(IndexSelection [illegible]);
}
```

```
package mastercatalog.gui;

import mastercatalog.ds.*;
import mastercatalog.gui.event.*;
import mastercatalog.util.*;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;

/**
  The renderer of a family sequence. Single clicking on a module selects it,
  double clicking triggers opening of the corresponding family frame.
  @author Lukas Knecht
*/

/*
  $Id: FamilySequenceRenderer.java,v 1.8 1999/08/19 11:25:45 knecht Exp $
  $Log: FamilySequenceRenderer.java,v $
  Revision 1.8  1999/08/19 11:25:45  knecht
  Removed drawModule().

  Revision 1.7  1999/08/10 12:55:11  knecht
  Change for new busy indicator.

  Revision 1.6  1999/07/30 08:43:13  knecht
  Various GUI changes and modifications.

  Revision 1.5  1999/07/19 08:16:29  knecht
  Various GUI changes.

  Revision 1.4  1999/04/26 10:13:19  knecht
  Modifications for V2.0

  Revision 1.3  1999/02/18 21:45:02  knecht
  Various big changes and enhancements for V1.0

  Revision 1.2  1998/12/16 16:24:45  knecht
  Added getFGColor() and getBGColor().

  Revision 1.1  1998/10/02 14:15:46  knecht
  Initial revision
*/

public class FamilySequenceRenderer extends JComponent
implements TableCellRenderer, ClickableCellRenderer {
  private CellState toDraw, clicked;
  private MCTable clickedTable;
  private int clickedRow;
  private Font font;
  private SlowAction slowAction;
  private IndexSelection selection;

  class CellState {
    int famId, indent, maxLength;
    FamilySequence sequence;
    Dimension size;

    CellState() { size = new Dimension(); }

    CellState(SequenceDisplayable displayable, int row, FamilySequence sequence,
              Dimension size) {
      this.sequence = sequence; this.size = size;
      setDisplayable(displayable, row);
    }

    void setDisplayable(SequenceDisplayable displayable, int row) {
      famId = displayable.getFamilyId();
      maxLength = displayable.getMaxLength();
      indent = displayable.getIndent(row);
```

```
    }

    // scale (0,maxLength) to (0,size.width)
    int scaleX(int x) { return x * size.width / maxLength; }

    // scale (0,10) to (0,size.height)
    int scaleY(int y) { return y * size.height / 10; }

    // find a member by its x coordinate
    Module findModule(int x) {
      int i;
      for (i = 0; i < sequence.module.length &&
             (scaleX(indent + sequence.module[i].seqStart) > x ||
              scaleX(indent + sequence.module[i].seqStart +
                     sequence.module[i].seqLength) <= x);
           i++);
      if (i < sequence.module.length) return sequence.module[i];
      else return null;
    }
  }

  /**
   Construct a new renderer for family sequences.
   */
  public FamilySequenceRenderer() {
    setBackground(Color.white); setFontSize(10);
    toDraw = new CellState();
    openAction = new SlowAction() {
      public void act() {
        Module clickedModule = clicked.findModule(clickedX);
        if (clickedModule != null) {
          FamilyFrame frame = FamilyFrame.openFrame
            (clickedTable.getServerConnection(), clickedModule.famId, null);
          if (frame != null) frame.appendToHistory();
        }
      }
    };
  }

  /**
   Set the font size used in rendering the modules.
   */
  public void setFontSize(int size) {
    font = Fontbase.getSansSerif(     , size);
  }

  /**
   Set the IndexSelection determining the selected modules.
   @param selection the IndexSelection belonging to the family displayed
   by this renderer or null if there are no selected modules.
   */
  public void setSelection(IndexSelection selection) {
    this.selection = selection;
  }

  public void paint(Graphics g) {
    if (toDraw.sequence != null) {
      getSize(toDraw.size);
      if (toDraw.sequence.wasModularized) g.setColor(Color.gray);
      else g.setColor(Color.lightGray);
      g.drawRect(toDraw.scaleX(toDraw.indent), toDraw.scaleY(4),
                 toDraw.scaleX(toDraw.sequence.length), toDraw.scaleY(2));
      g.fillRect(toDraw.scaleX(toDraw.indent), toDraw.scaleY(4),
                 toDraw.scaleX(toDraw.sequence.length), toDraw.scaleY(2));
      if (font != null) g.setFont(font);
      for (int i = 0; i < toDraw.sequence.module.length; i++) {
        Module module = toDraw.sequence.module[i];
        boolean ofFamily = toDraw.famId == module.famId;
        DrawingUtilities.drawModule
          (g,
           toDraw.scaleX(toDraw.indent + module.seqStart),
           toDraw.scaleY(1),
```

```
            toDraw.scaleX(module.seqLength),
            toDraw.scaleY(9),
            module.famName, ofFamily,
            ofFamily && selection != null &&
            selection.getSelected().get(module.index));
        }
        if (!toDraw.sequence.wasModularized) {
          String text = "                    ";
          FontMetrics fm = g.getFontMetrics();
          int w = fm.stringWidth(text);
          int x = toDraw.scaleX(toDraw.indent) +
            (toDraw.scaleX(toDraw.sequence.length) - w) / 2;
          g.setColor(Color.black);
          g.drawString(text, x, toDraw.scaleY(8));
        }
      }
    }

    /** The TableCellRenderer implementation: **

    public Component getTableCellRendererComponent(JTable      , Object      ,
                                                   boolean     ,
                                                   boolean     ,
                                                   int     , int     ) {
      toDraw.setDisplayable((SequenceDisplayable)table.getModel(), row);
      toDraw.sequence = (FamilySequence)value;
      return this;
    }

    * The ClickableCellRenderer implementation: **

    public void cellClicked(MCTable     , CellClickEvent     , Object     ) {
      if (value != null)
        if (event.getClickCount() == 1 && selection != null) {
          CellState state = new CellState((SequenceDisplayable)table.getModel(),
                                          event.getRow(),
                                          (FamilySequence)value,
                                          event.getSize());
          Module module = selected.findModule(event.getX());
          if (module != null && module.famId == selected.famId) {
            selection.toggle(module.index, event.isControlDown());
            selection.propagate();
          }
        }
        else if (event.getClickCount() == 2) {
          clicked = new CellState((SequenceDisplayable)table.getModel(),
                                  event.getRow(),
                                  (FamilySequence)value, event.getSize());
          clickedTable = table; clickedX = event.getX(); openAction.run();
        }
    }

    * Get position dependent tooltips: **

    public String getToolTipText(MCTable     , CellClickEvent     ,
                                 Object     ) {
      String text = null;
      if (value != null) {
        CellState state = new CellState((SequenceDisplayable)table.getModel(),
                                        event.getRow(),
                                        (FamilySequence)value, event.getSize());
        Module module = state.findModule(event.getX());
        if (module != null) {
          text = "                " + module.famName;
          if (module.famId == state.famId) text = text +          ;
          else text = text +                                      ;
        }
      }
      return text;
    }
}
```

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
22 March 2001 (22.03.2001)

PCT

(10) International Publication Number
WO 01/20535 A3

(51) International Patent Classification⁷: G06F 19/00

(21) International Application Number: PCT/US00/25247

(22) International Filing Date:
14 September 2000 (14.09.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/154,149	14 September 1999 (14.09.1999)	US
09/397,335	14 September 1999 (14.09.1999)	US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier applications:

US	60/154,149 (CIP)
Filed on	14 September 1999 (14.09.1999)
US	09/397,335 (CIP)
Filed on	14 September 1999 (14.09.1999)

(71) Applicant *(for all designated States except US)*: ERAGEN BIOSCIENCES, INC. [US/US]; 12085 Research Drive, Alachua, FL 32615 (US).

(72) Inventors; and

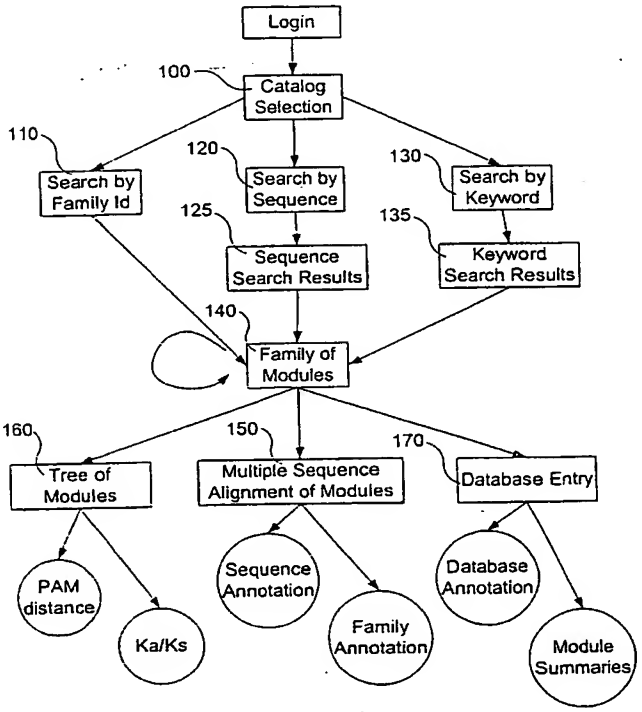
(75) Inventors/Applicants *(for US only)*: CHAMBERLIN, Stephen [GB/US]; 12085 Research Drive, Alachua, FL 32615 (US). BENNER, Steven, A. [US/US]; 1501 NW 68th Terrace, Gainesville, FL 32605 (US). KNECHT, Lukas [CH/US]; Neptunstrasse 35, CH-8032 Zurich (CH).

(74) Agents: McLEOD, Christine, Q. et al.; Saliwanchik, Lloyd & Saliwanchik, 2421 N.W. 41st Street, Suite A-1, Gainesville, FL 32606 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ,

[Continued on next page]

(54) Title: GRAPHICAL USER INTERFACE FOR DISPLAY AND ANALYSIS OF BIOLOGICAL SEQUENCE DATA



(57) Abstract: A computer research tool is provided for searching and displaying biological data. Specifically, the invention provides a computer research tool for performing computerized research of biological data from various databases and for providing a novel graphical user interface that significantly enhances biological data representation, progressive querying and cross-navigation of windows and databases. The invention can be implemented in numerous ways, including as a system, a device, a method, or a computer readable medium.

WO 01/20535 A3

NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) **Designated States** *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

(88) **Date of publication of the international search report:**
17 January 2002

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTERNATIONAL SEARCH REPORT

Inte ional Application No
PCT/US 00/25247

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F19/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F G06T

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	PERRIERE G ET AL: "WWW-QUERY: AN ON-LINE RETRIEVAL SYSTEM FOR BIOLOGICAL SEQUENCE BANKS" BIOCHIMIE,PARIS,FR, vol. 78, 1996, pages 364-369, XP001014183 ISSN: 0300-9084 the whole document ---	1-31
Y	WO 98 26407 A (INCYTE PHARMA INC) 18 June 1998 (1998-06-18) abstract; claim 1 ---	1-31
Y	PAGE RDM: "TreeView: An application to display phylogenetic trees on personal computers" COMPUTER APPLICATIONS IN THE BIOSCIENCES, 1996, pages 357-358, XP001010348 the whole document --- -/--	1-31

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
19 July 2001	26/07/2001

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Filloy García, E

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Inte. .ional Application No
PCT/US 00/25247

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	YOSHIKAWA T ET AL: "ON THE IMPLEMENTATION OF A PHYLOGENETIC TREE DATABASE" VICTORIA, BC, AUG. 22 - 24, 1999,NEW YORK, NY, IEEE,US, vol. CONF. 7, 22 August 1999 (1999-08-22), pages 42-45, XP000898889 ISBN: 0-7803-5583-0 the whole document ---	1-31
A	EERNISSE D J: "A brief guide to phylogenetic software" TRENDS IN GENETICS,NL,ELSEVIER SCIENCE PUBLISHERS B.V. AMSTERDAM, vol. 14, no. 11, 1 November 1998 (1998-11-01), pages 473-475, XP004146861 ISSN: 0168-9525 -----	

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int. ..ional Application No
PCT/US 00/25247

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9826407 A	18-06-1998	US 5966712 A	12-10-1999
		WO 9826408 A	18-06-1998
		US 5970500 A	19-10-1999